12-2016

Security techniques for sensor systems and the Internet of Things

Daniele Midi
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Daniele Midi

Entitled
Security Techniques for Sensor Systems and IoT

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Elisa Bertino Chair	Cristina Nita-Rotaru
Sonia Fahmy	
Mathias Payer	
Dongyan Xu	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Elisa Bertino

Approved by: Sunil Prabhakar / William J. Gorman 9/12/2016

Head of the Departmental Graduate Program Date

SECURITY TECHNIQUES FOR SENSOR SYSTEMS AND THE INTERNET OF THINGS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Daniele Midi

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Midi, Daniele PhD, Purdue University, December 2016. Security Techniques for Sensor Systems and the Internet of Things. Major Professor: Elisa Bertino.

Sensor systems are becoming pervasive in many domains, and are recently being generalized by the Internet of Things (IoT). This wide deployment, however, presents significant security issues.

We develop security techniques for sensor systems and IoT, addressing all security management phases. Prior to deployment, the nodes need to be hardened. We develop nesCheck, a novel approach that combines static analysis and dynamic checking to efficiently enforce memory safety on TinyOS applications. As security guarantees come at a cost, determining which resources to protect becomes important. Our solution, OptAll, leverages game-theoretic techniques to determine the optimal allocation of security resources in IoT networks, taking into account fixed and variable costs, criticality of different portions of the network, and risk metrics related to a specified security goal.

Monitoring IoT devices and sensors during operation is necessary to detect incidents. We design Kalis, a knowledge-driven intrusion detection technique for IoT that does not target a single protocol or application, and adapts the detection strategy to the network features. As the scale of IoT makes the devices good targets for botnets, we design Heimdall, a whitelist-based anomaly detection technique for detecting and protecting against IoT-based denial of service attacks.

Once our monitoring tools detect an attack, determining its actual cause is crucial to an effective reaction. We design a fine-grained analysis tool for sensor networks that leverages resident packet parameters to determine whether a packet loss attack is node- or link-related and, in the second case, locate the attack source. Moreover,

we design a statistical model for determining optimal system thresholds by exploiting packet parameters variances.

With our techniques' diagnosis information, we develop Kinesis, a security incident response system for sensor networks designed to recover from attacks without significant interruption, dynamically selecting response actions while being lightweight in communication and energy overhead.

1 INTRODUCTION

In the current data-driven world, sensors and sensor networks are becoming pervasive in a large number of application domains. Their small size, low cost, and limited need for resources are among the main factors of their widespread use. At home, smart thermostats and intelligent refrigerators reduce energy consumption and increase user comfort. In manufacturing, autonomous controllers operate and monitor complex production pipelines. In agriculture, large-scale networks of small sensors collect information about the environment and enable "precision" agriculture.

More recently, the notion of pervasive sensing and computing has been generalized by the Internet of Things (IoT). IoT merges the benefits of smart embedded systems with the power of connected Internet-based services, computation, and management. Different categories of IoT devices are emerging, with capabilities ranging from automatic data acquisition, to control, to networking [1]. It is thus clear that in the near future, as well as in the longer term, we will increasingly see a pervasive deployment of different types of computing devices connected by different communication mechanisms. Mckinsey & Company estimates the economic impact of IoT by 2025 will range from $2.7 to $6.2 trillion dollars [2]. In addition, Gartner 2015 forecast states that by the year 2020 we will see 20.8 billion IoT devices installed [3].

However, with the widespread use of sensor systems and IoT, security must become a first-class citizen [4–7]. Most applications – e.g., mission-critical tasks, industrial control or medical monitoring – have stringent requirements with respect to end-to-end system reliability, trustworthy data delivery, and service availability. Sensor systems and IoT devices are often resource-constrained, communicate via an unreliable wireless medium, operate in unattended environments, and usually lack any tamper-proof packaging. These conditions make the network nodes vulnerable to operational failures. Moreover, the insecure and vulnerable nature of sensor envi-

ronments make them vulnerable to attacks that falsify context, modify access rights, and, in general, disrupt the system operation [8]. Malicious attacks can have a wide range of consequences, from the malfunctioning of smart city sensors and controllers, to a hospital patient receiving the wrong treatment in a smart healthcare scenario. Several attacks to embedded devices, sensor systems and IoT have been recently reported [9–12]. Successful remote hacks on critical life devices, such as insulin pumps and pacemakers, by exploiting their insecure wireless communications [13] raise also critical safety issues on the use of interconnected sensors and actuators. A recent study by HP about the most popular devices in some of the most common IoT niches revealed an alarmingly high average number of vulnerabilities per device [14]. On average, 25 vulnerabilities were found per device. For example, 80% of devices failed to require passwords of sufficient complexity and length, 70% did not encrypt local and world traffic communications, and 60% contained vulnerable user interfaces and/or vulnerable firmware [14].

IoT networks can differ from each other for several aspects, such as topology, mobility, size, degree of heterogeneity, location, communication modality, and so on. However, they also have some common characteristics, namely the use of devices with low computational power and low energy consumption. They are also prone to physical attacks and eavesdropping, since they are often unattended and typically communicate via wireless channels.

Our work, presented in this dissertation, aims at developing security techniques for sensor systems and the IoT. It is important to note that even though security for IoT is a much less mature area than security for sensor networks, many existing security techniques developed for sensor networks can be directly applied to the more general setting of IoT. However, other security techniques need significant extensions because of the heterogeneity of communication mechanisms, platforms, configurations, and specific security requirements, which opens an extremely wide attack surface, while at the same time increases the difficulty of deploying all-encompassing security solutions. Moreover, unlike wireless sensor networks (WSNs), IoT devices are susceptible

not only to attacks from other devices in the network, but also from more powerful attackers from the untrusted Internet.

The design of our security techniques is based on a security lifecycle consisting of four phases: Prepare and Prevent; Monitor and Detect; Diagnose and Understand; React, Recover, and Fix. Each phase is addressed by our work with one or more projects, and this dissertation presents the research results achieved in each of those projects. We design all of our techniques to be able to work in concert, addressing the different aspects of the security lifecycle of sensor systems and IoT, and our results show that each one is effective in fulfilling its role in the overall security enforcement scenario.

Thesis Statement. *The intrinsic characteristics of the sensor and IoT domain expand the attack surface of computer and communication systems. Existing security techniques need to be analyzed, extended, and modified in order to efficiently and effectively achieve security across heterogeneous and constrained scenarios, throughout all the four phases of hardening, monitoring, diagnosing, and recovering.*

In the remainder of this introduction, we present the four identified security lifecycle phases and the approaches we developed to address them, as well as present the overall vision of how all the techniques work together to achieve the end goal of security for sensor systems and IoT.

1.1 Security Lifecycle Phases and Developed Techniques

Prepare and Prevent. The first step in securing a system consists of hardening the system itself before its deployment, as well as deploying measures to prevent attacks. For both sensor systems and IoT, this includes leveraging techniques able to protect the software installed on the devices from unintentional bugs and vulnerabilities that an attacker could exploit. In Chapter 2, we present nesCheck [15], a

novel approach that combines static analysis and dynamic checking in order to efficiently enforce memory safety on existing embedded software, without requiring any source modification. nesCheck targets TinyOS, one of the most popular embedded operating systems. We tailor our approach to leverage the peculiar characteristics of sensor systems, which make existing memory safety solutions not suitable but which offer interesting opportunities for new, highly efficient techniques. nesCheck analyzes the source code, identifies the conservative set of vulnerable pointer variables, finds static memory bugs, and instruments the code with the appropriate dynamic runtime checks. We implemented a prototype of nesCheck extending the existing TinyOS compiler toolchain with custom tools built on the LLVM compiler suite. Extensive evaluation, on stress-intensive programs as well as standard TinyOS applications, proves that nesCheck effectively enforces memory protection, while minimizing the number of dynamic runtime checks required and thus the performance impact. nesCheck reduces the attack surface for malicious adversaries, as well as protects the availability and integrity of the system from unintentional bugs. Therefore, it is a fundamental building block in our security efforts for sensor systems and IoT, by also protecting our other security software systems – discussed later in this dissertation – from attackers. As compared to existing work [16–18], nesCheck is specifically designed for the constraints, challenges, and advantages of embedded devices, does not require source code modifications or annotations, and leverages more extensive static analysis techniques to conservatively minimize the performance overhead.

As an additional step when deploying security measures in a sensor and IoT network, deciding the best placement for the various security resources plays an important role. Different portions of the network and different devices have different levels of importance in achieving the overall security goals. Protections such as that offered by nesCheck come at a cost, and determining the optimal allocation plan for security measures must take into account the available security resources and their capabilities, their fixed cost, and runtime energy consumption, how critical different areas of the network are, as well as the risk associated with successful attacks

on them. We design and implement OptAll [19], a game-theory-based method to compute the optimal security resource allocation plan through a Pareto optimization problem. We present OptAll in Chapter 3 of this dissertation. Prior approaches in this domain targeted homogeneous, traditional computing deployments [20, 21], or a single specific attack [22–24], and are focused on improving performance [25]. To the best of our knowledge, OptAll is the first work to address the problem of finding the optimal security resource allocation plan for IoT networks, taking into account fixed and variable costs, criticality, and risk. Our evaluation on several network topologies – both grid-based and random – shows that OptAll is able to determine the most efficient and effective allocation plan for security resources with respect to any specified security goal.

Monitor and Detect. During their operation, IoT devices and sensors must be continuously monitored in order to detect anomalies, attacks, and operational failures. While much research has been carried out in the last decade on Intrusion Detection Systems (IDSes), very few solutions are targeted to sensor systems and IoT. These approaches have several drawbacks, such as requiring invasive software modifications to the devices' firmware, being limited to only a single device or group of devices, not supporting interoperation, and delegating security to the individual manufacturers [26–29]. Moreover, the simple adaptation of existing IDSes, designed for traditional computing systems and networks, is not viable, since widely-adopted approaches – such as full network scanning – are not viable for the IoT. Several characteristics of IoT, however, can be leveraged to design an IDS well fit for this domain. For example, while the communication protocols and mediums are very heterogeneous, they are mostly standardized, enabling effective use of promiscuous overhearing and watchdog-based mechanisms. Moreover, most devices and IoT network have specific features (e.g., single-hop vs. multi-hop topology, mobility, ...) that can help in ruling out attacks, removing ambiguity, and improving detection accuracy. We thus first investigate the relationships between IoT network features and related

attacks, and then design and develop Kalis [30], a self-adapting, knowledge-driven IDS for IoT able to detect attacks in real time across IoT systems running different communication protocols and with different security goals. Kalis autonomously collects knowledge about the features of the monitored network, and leverages such knowledge to dynamically configure the most effective set of detection techniques. Chapter 4 of this dissertation presents this system. To the best of our knowledge, Kalis is the first comprehensive approach to intrusion detection for IoT that does not target an individual protocol or application, and adapts the detection strategy to the specific network features.

While Kalis is able to defend against threats to the IoT network and its devices, another challenging type of attack is represented by distributed denial of service (DDoS) attacks via botnets. The large amount of IoT devices and sensors that are expected to be deployed in the near future makes the IoT an ideal vector for DDoS attacks via botnets. Chapter 5 of this dissertation presents our work on this topic. We analyzed the actual attack power that small IoT devices have when used as part of a botnet. Our results show that they have a very high potential for a DDoS attack. We thus design and develop Heimdall [31], a whitelist-based anomaly detection technique tailored to IoT devices. Our technique operates on routers acting as gateways and is effective in identifying and blocking DDoS attacks by IoT botnets. Existing approaches target traditional computing networks rather than IoT systems, and leverage communication patterns [32] or artificial immune system-based techniques [33,34] to determine normal behaviors. Heimdall explicitly targets IoT botnets, and its contributions include leveraging the IoT intrinsic characteristics to design simpler but more accurate techniques to perform anomaly detection tasks.

Diagnose and Understand. In the security lifecycle, once an attack has been detected, it is critical to perform a diagnosis of the attack to determine the actual cause of the attack. Such diagnosis is crucial in order to properly respond to the attack. Packet losses in sensor networks are a particularly relevant class of attacks

and can be caused by either attacks affecting the nodes – e.g., selective forwarding or blackhole attacks – or attacks focusing on the wireless links – the introduction of interference in the wireless medium, when wireless networks are used for communication. Chapter 6 of this dissertation presents the design and implementation of our fine-grained analysis (FGA) tool for WSNs [35], a fully distributed, event-driven solution that leverages resident packet parameters such as RSSI and LQI to determine the most likely cause of a packet loss event and, in case of interference, even to accurately locate the source of jamming. Real-world testbed experiments show that our FGA tool is effective in differentiating between the various attacks that may affect nodes and links. Related approaches mostly focus on detecting packet losses rather than the important topic of the cause of the loss [36–39]. Moreover, our FGA tool relies on a smaller set of network parameters and focuses on differentiating node- and link-related attacks rather than natural losses and malicious discardings [40, 41].

The accuracy of the FGA tool relies on the correct choice of some system parameters and thresholds, and empirically-determined values might not always be optimal. Moreover, the choice of a single threshold value for the entire network, can be suitable for some neighborhoods but not appropriate for others. Therefore, we designed an approach that builds a statistical model for determining optimal system thresholds by exploiting the variances of RSSI and LQI [42]. This statistical approach also has the advantage of allowing an individual threshold for each link. In Chapter 7 we present such a model, together with its validation through extensive MATLAB simulations based on real sensor data, showing that our model is accurate and its system parameters lead to an optimally-accurate fine-grained analysis of the underlying causes of packet losses.

React, Recover and Fix. The applications running on sensor systems and IoT often impose stringent requirements on data reliability and service availability, due to the deployment of sensor networks in various critical infrastructures. Given the failure- and attack-prone nature of sensor networks, enabling them to continu-

ously provide their services as well as to effectively recover from attacks is a crucial requirement. The accurate diagnosis information provided by our previously introduced solutions allow one to undertake more effective response actions. Chapter 8 presents Kinesis [43], a security incident response system for WSNs designed to keep the network functional despite anomalies or attacks and to recover from attacks without significant interruption. Kinesis is quick and effective in responding to incidents, distributed in nature, dynamic in selecting response actions based on the context, and lightweight in terms of response policy specification and communication and energy overhead. A per-node single timer-based distributed strategy to select the most effective response executor in a neighborhood makes the system simple and scalable, while achieving load balancing and redundant action optimization. The contributions over related systems include a wide range of response actions that go beyond simply rerouting data or isolating the misbehaving node [29, 44], no need for a centralized node to manage reputations and choose response agents [45], and a lightweight architecture designed for constrained systems [46]. Extensive simulations and testbed experiments show that Kinesis successfully counteracts anomalies/attacks and behaves consistently under various attack scenarios and rates.

1.2 Overall Functional Framework

With the integration of the techniques we developed, we envision an overall functional framework that provides security to sensor systems and IoT systems in concert throughout all the four phases of security lifecycle (see Figure 1.1.)

The use of OptAll guides the initial phase of security planning. It provides an optimal allocation plan for the provisioning of the other security techniques we developed. For a set of nodes in the network, the memory safety guarantees provided by nesCheck is crucial in maintaining the network functional. Choosing strategically which nodes to protect can guarantee an optimal trade-off between performance, cost, and security. The deployment of Heimdall on the IoT gateway ensures that

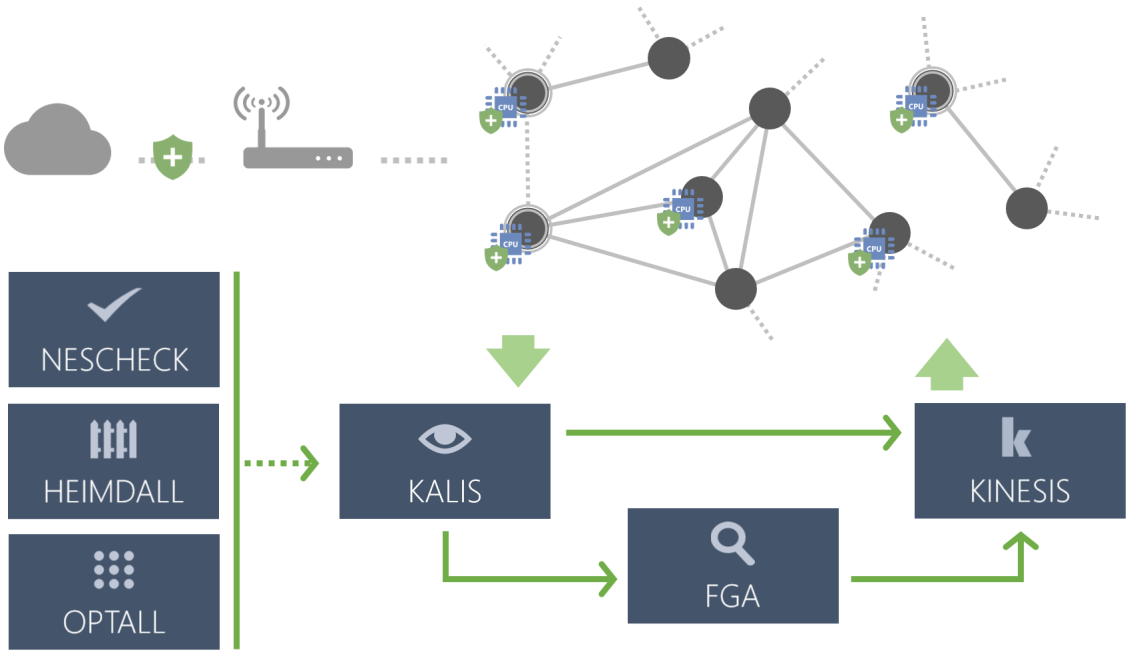


Figure 1.1. The overall functional framework integrating the techniques developed in this dissertation.

the local devices are more protected against enrollment in botnets, and prevents any compromised local device from partaking in DDoS attacks towards remote targets. The knowledge about these prevention and hardening mechanisms is an essential input information to Kalis, our IDS, and OptAll can once again guide the decision of the exact locations in which to deploy the IDS nodes. Kalis continuously monitors the network and collects dynamic knowledge about the features of the monitored network and entities, always selecting the best set of detection techniques out of its extensive library. Once an attack or anomaly has been detected, Kalis notifies Kinesis to take the most appropriate response action(s) to react with respect to the overall chosen security goal. In cases of ambiguity on the potential attack(s) detected, our Fine-Grained Analysis tool can investigate further into the incident in order to allow a more informed decision by Kinesis on the response action to take. Since the monitoring and reaction are continuous and in real-time, the security enforcement

can maintain the network and its devices functional in face of subsequent security incidents and disruptions.

2 NESCHECK: STATIC ANALYSIS AND DYNAMIC INSTRUMENTATION FOR NESC MEMORY SAFETY

In the overall scenario of the security lifecycle we target, the first step towards protecting the deployment of a sensor system is that of hardening the system to shrink the attack surface and prevent incidents. With the deployment of WSNs for sensitive, real-time applications, availability and integrity are of paramount importance. Moreover, as WSN nodes often manage confidential information – such as private keys and aggregated data – confidentiality and integrity also become key requirements. However, the distributed and concurrent nature of WSN applications, together with the intrinsic type and memory unsafety of C/C++, make it hard to achieve these security goals.

TinyOS [47] is an open source operating system designed for low-power wireless embedded systems, such as WSN motes and smart meters [48]. TinyOS programs consist of separate software components statically linked through interfaces. Common components include routing and packet radio communication, sensor measurements, and storage. The language used to program TinyOS applications is *nesC*, a dialect of the C language optimized for the resource constraints of low-power embedded devices [49]. Because of the strict constraints in terms of memory, storage, and energy, neither TinyOS nor the underlying hardware provide any memory protection or virtual memory mechanism between processes or kernel and user-space. Moreover, the *nesC* language makes it easy to write memory-unsafe code, inheriting all the type and memory safety problems of C.

Memory corruption in the software running on a single node may allow an attacker to take over the node, read private data, or even disseminate incorrect data and degrade the entire network. Note that embedded platforms do not have code injection protection or ASLR, so a holistic defense like memory safety becomes even

more important. More critically, since all the nodes run the same software image, an attacker may exploit a single vulnerability to take control of every node in the network. Concrete examples of such devastating attacks have been shown for Harvard-architecture-based sensor nodes such as the MicaZ motes [50], as well as Von Neumann-architecture-based ones such as the popular TelosB motes [51]. In these attacks, a well-crafted network packet sent to a vulnerable node can take control of the node and propagate as a self-replicating worm to the entire network through multi-hop communications [52–56]. All of these critical attacks would be prevented by the enforcement of memory safety.

Existing memory safety techniques do not apply to embedded systems, nesC, or TinyOS, as they are designed for general-purpose systems. Embedded systems fundamentally differ from regular computing systems. While general purpose systems provide plenty of ROM, memory and an MMU, on embedded systems, memory is scarce (but dedicated) and there is usually no MMU or even distinction between kernel-space and user-space. For example, widespread sensor motes like the Memsic TelosB [57] only provide 10kB of RAM and 48kB of program flash memory; previously proposed memory safety approaches result in significantly bigger code size and more intensive memory usage. Moreover, the performance degradation that many existing solutions impose is not acceptable for energy-constrained, real-time WSNs applications. In fact, solutions such as CCured reported slowdowns ranging from 20% to 200% [16]. Given the resource constraints, straightforward porting of common implementations for memory protection techniques to embedded systems is infeasible. A tailored solution for memory and type safety for TinyOS applications is therefore needed.

For TinyOS applications, the code for applications, libraries, and operating system is entirely available at compile time. This allows to effectively employ whole-program static analysis and dynamic checking instrumentation to ensure memory safety. Moreover, by statically identifying and removing unnecessary checks for memory accesses that will never result in memory errors, it is possible to achieve a low performance

overhead. Based on such considerations, we design *nesCheck*, a novel scheme that tailors static analysis, type inference, and dynamic instrumentation techniques to embedded applications to enforce memory safety on existing nesC programs. The goal of nesCheck is to protect embedded software against memory vulnerabilities with negligible overhead and without requiring any source code modification. nesCheck statically analyzes the source code, identifies the potentially dangerous pointer variables, automatically infers the minimum set of dynamic runtime checks needed to enforce memory safety based on pointer access flow, and instruments the code appropriately.

nesCheck is novel in bringing memory safety to embedded devices by adapting instrumentation-based approaches to the challenges and advantages of the embedded world, where whole-program analysis is feasible but memory and performance overhead are major concerns.

To evaluate our approach, we implement nesCheck as a combined static analysis/-dynamic checker on top of the LLVM compiler framework. The static analysis infers types and removes as many checks as possible while the dynamic checker enforces safety. We then integrate our checker into the existing nesC toolchain. We evaluate nesCheck on standard TinyOS application benchmarks, and show that it effectively enforces memory safety on WSN applications, while minimizing the runtime performance overhead (0.84% on energy, 5.3% on code size, up to 8.4% on performance, and 16.7% on RAM). These benchmarks are the standard benchmarks for evaluating WSN and present realistic usage scenarios for embedded systems.

The contributions of our work are:

- Design of an inter-procedural whole-program static analysis mechanism, based on type tracking and pointer usage, and without the need for programmer annotations;

- Design of dynamic instrumentation for efficient memory safety enforcement on highly constrained embedded platforms, without MMU or kernel/user space separation;
- Evaluation of the efficiency and effectiveness of our approach through an full implementation prototype.

2.1 Adversarial Model

We assume that the attacker can inject and intercept arbitrary packets in the network. We also assume that the application has memory vulnerabilities known to the attacker. She will exploit them to take control of a node by means of code injection/reuse attacks (following the intrinsics of the underlying hardware), compromising the integrity, availability, and confidentiality of the node.

Physical attacks targeting the nodes, in which the adversary tampers with the hardware of a mote and/or directly reprograms the node with malicious firmware, are out of the scope of nesCheck.

2.2 Background

2.2.1 Memory Safety Vulnerabilities

The root cause of all memory safety vulnerabilities is the dereferencing of invalid pointers. There are two main categories of memory safety vulnerabilities: *spatial* memory safety vulnerabilities, resulting from pointers pointing to addresses outside the bounds of the allocated memory area, and *temporal* memory safety vulnerabilities, resulting from the usage of pointers after the corresponding memory areas are deallocated (e.g. *use-after-free* errors).

Our current prototype of nesCheck targets spatial memory safety, but can be extended to enforce temporal safety as well, by lock and key mechanisms [58]. However, as memory in well-developed WSN applications is allocated statically instead of

dynamically, temporal safety errors are not an important issue for applications that comply with the development guidelines for TinyOS. This includes all the applications that ship with the standard distribution of TinyOS, as well as most larger-scale WSN applications. Examples of the memory vulnerabilities that nesCheck protects against are out-of-bounds accesses to pointers on the stack and heap, uninitialized uses, and null dereferencing.

2.2.2 TinyOS

nesC. nesC is an event-driven dialect of C. Its additional features include the concept that programs are built out of *components*, statically linked through interfaces.

Dynamic allocation. In the early versions of TinyOS, dynamic memory allocation was not allowed. This constraint, partially relaxed in recent releases, is still highly discouraged, as the lack of memory protection and separation can easily lead to involuntary stack smashing when the heap grows into the stack [59]. Specialized components (e.g. TinyAlloc), were introduced to provide support for dynamic allocation, but behind the scenes they are however still simply managing a large chunk of pre-allocated memory. Disabling dynamic allocation has the advantage, from a memory safety standpoint, that most needed information is available at compile-time, and little is left for dynamic detection.

Compilation and execution model. The standard TinyOS compilation pipeline is composed of several steps. First, the nesC code is processed and all the required components, including the operating system, are linked together. Under this model, all code, libraries, and OS components are statically known at compile time. The resulting single nesC code is cross-compiled to C code, in turn compiled natively into a binary image for the specific target platform. Such single binary image – containing both user code and OS code – runs as a single executable, assuming complete control over the hardware at all times. The memory address space is shared among all components, both user and system code. For this reason, the official development

guidelines for TinyOS recommend to (i) keep the state of the various components private, (ii) communicate only through exposed interfaces, and (iii) avoid transferring pointers between different pieces of code. All these characteristics of the TinyOS compilation and execution model make it a particularly good fit for static analysis.

2.3 The nesCheck Approach

Figure 2.1 shows the architecture of the final pipeline for nesCheck, and our main memory safety goals are listed below. nesCheck performs both static bug detection – for memory accesses that will always result in a violation regardless of the execution path – and runtime bug catching – for memory accesses that could potentially lead to memory corruptions, depending on the execution flow.

Bugs: (Static) Find all *statically provable* memory bugs and report them as errors;

Vulnerabilities: (Static) Find all *potentially* unsafe memory accesses, determine and exclude those that will never result in a memory corruption (in a conservative way), and report the remaining ones as warnings;

Checks: (Dynamic) Instrument all remaining vulnerable locations with dynamic runtime checks, and catch all memory errors at runtime.

2.3.1 Static Analysis

nesCheck uses static analysis in order to enforce an extended type system on the pointer variables, and subsequently compute and propagate metadata for the vulnerable pointers. Our approach uses an inter-procedural whole-program analysis technique, carried out on the Static Single Assignment (SSA) form [60] representation of the code. In SSA form, each variable is written to at most once, introducing a fresh variable every time the value is updated with a destructive assignment operation. While the code is in SSA form, the heap remains in non-SSA form, meaning that the same memory location can be written to multiple times through the same and different pointers. Therefore, even though in SSA form each variable is only assigned once, a

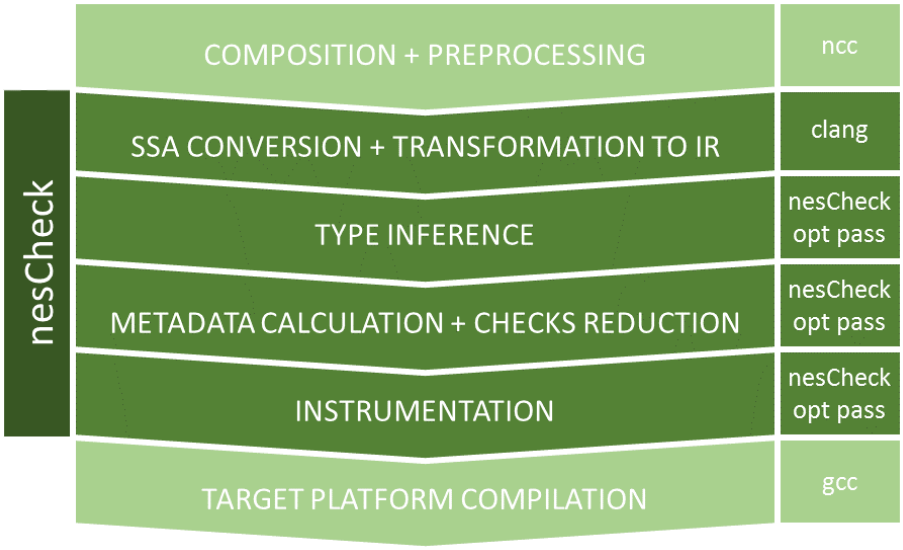


Figure 2.1. The complete nesCheck pipeline, with lighter blocks being existing steps of the nesC compiler toolchain, and darker blocks the newly introduced ones.

new value is assigned with a `store` operation to a memory location previously loaded with a `load` operation, making it possible to connect together different instructions operating on the same logical variable.

Extended Type System and Type Inference

In order to provide type safety, identify the potentially dangerous memory accesses, and avoid dynamic checks on the provably safe operations, it is necessary to understand the role played by the various pointers in the code and their interrelations. We thus enforce a type system inspired by CCured [16], that categorizes pointers according to their usage into different classes with specific characteristics. The pointer types that we consider are the following: **(i) Safe pointer to τ:** it can only be null or point to a value of type τ. At runtime, it may only need a null-pointer check. **(ii) Sequence pointer to τ:** like a Safe pointer, it can be null or point to a value of type τ. However, a Sequence pointer can also be interpreted as an integer, and be

ALGORITHM 1: nesCheck's type inference algorithm

```
foreach declaration of pointer variable p do
    classify(p, SAFE);
foreach instruction I using pointer p do
    r ← result_of(I);
    if I performs pointer arithmetic then
        classify(p, SEQ);
        classify(r, SAFE);
    if I casts p to incompatible type then
        classify(p, DYN);
        classify(r, DYN);
```

manipulated via pointer arithmetic. At runtime, it may need a null-pointer check, as well as a bounds check if cast to a safe pointer of base type τ. **(iii) Dynamic pointer:** it is a pointer that cannot be statically typed. At runtime, it may need null-pointer, bounds, and dynamic type checks.

The type inference engine gathers information from the source code to classify pointer declarations according to the extended type system. The engine focuses on all locations in which pointer variables are used and classifies them, in a fixpoint iteration, by analyzing their usage. Our type inference algorithm is shown in Algorithm 1.

The type inference algorithm uses 3 rules:

① All pointers are classified as *Safe* upon their declaration.

② Safe pointers subsequently used in pointer arithmetic are re-classified as a *Sequence*.

③ Safe or Sequence pointers interpreted with different types in different locations are re-classified as a *Dynamic*. This includes casting between different levels of indirection (e.g., `int**` to `int*`), and between different root types (e.g., `int*` to `void*`).

nesCheck's type inference engine effectively enforces a total ordering $Dynamic \prec Sequence \prec Safe$ on pointer types, so the type of a pointer is updated only if the new type is more restrictive. For example, in the following code portion:

```
1 int *arr, *p, n;
2 arr = malloc(5 * sizeof(int));
3 n = (int)arr;
4 p = arr[3];
```

the pointer `*arr` is classified as Safe upon declaration. When casted from `int*` to `int`, `*arr` is reclassified as Dynamic since $Dynamic \prec Safe$ holds according to the total ordering. However, when used in pointer arithmetic, the type of `*arr` is not changed as the total ordering constraint $Sequence \prec Dynamic$ is not satisfied.

Note that no extra rules are necessary for some non-obvious cases, often because the analysis runs on SSA form. For instance, indirect calls (e.g., callbacks or function pointers) are classified as Dynamic by nesCheck's type inference because of the use of `void*` pointers. Another case includes pointers to pointers, or pointers to structs containing pointers. If the inner type is classified as Dynamic, the outer type must be classified as Dynamic as well. A concrete example of this is `int * q1 * q2`, where `q1` and `q2` are pointer kinds. If `q2` is Dynamic, the `q1` should also be Dynamic. The three rules presented suffice in correctly classifying these pointers, since an access to that pointer as a whole will result in two subsequent `load` instructions, that will propagate the Dynamic classification between the different levels of indirection.

After the type inference completes, all the pointers are classified. The rules guarantee that the final assignments are a conservative over-approximation, potentially classifying non-Dynamic pointers as Dynamic pointers, but never the opposite. This fundamental property ensures the correctness of the memory safety enforcement. The subsequent optimizations will compensate the potential performance degradation of conservative classification.

Type Inference Validation. To explicitly validate the accuracy of nesCheck's type inference engine, we walk through one example, picking the code originally presented in the CCured paper [16]:

```
1  int **a;
2  int i;
3  int acc;
4  int **p;
5  int *e;
6  acc = 0;
7  for (i=0; i<100; i++) {
8      p = a + i;
9      e = *p;
10     while ((int) e % 2 == 0)
11         e = *(int**)e;
12     acc += ((int)e >> 1);
13 }
```

The program sums an array of "boxed integers", a data type with double interpretation: when odd, its 31 most significant bits can be interpreted as an integer, otherwise it represents the pointer to another boxed integer.

The expected behavior of the type inference engine is to classify `**a` as Sequence – since it is used in pointer arithmetic at line 8 – and `*e` as Dynamic – since it is casted and used with different types at different locations (i.e., as pointer at line 11 and as integer at line 12). All the other pointers should be classified as Safe. It is possible to verify that, according to our type inference algorithm, the correct classification of `**a` as Sequence pointer is achieved by the application of Rule ②, while `*e` is correctly classified as Dynamic thanks to Rule ③ applied at line 12. All the other pointers are classified as Safe upon their declaration, by Rule ①, and never change their classification.

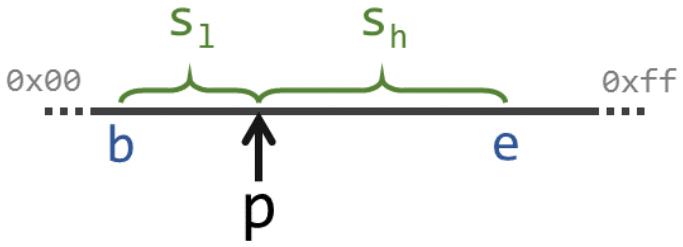


Figure 2.2. Comparison of bounds metadata in nesCheck vs. the traditional approach.

Through this example and others constructed specifically to exercise unusual pointer usages, we verify that nesCheck correctly classifies all of the pointer types according to our extended type system.

Metadata Computation and Propagation

The metadata maintained by nesCheck for each pointer contains information about the memory area to which such pointer points. Differently from the traditional tracking of base b and bound e for each pointer, nesCheck's metadata includes the size of the areas towards both lower and higher memory addresses (denoted with s_l and s_h, respectively), with respect to the current address stored in the pointer variable. Figure 2.2 shows a graphical comparison of our metadata structure and the more traditional one. As an example, let `int* p` be a pointer to an array of 5 integers, and assume `int* p1 = &p[2]`. The metadata for `p` will be ($s_l = 0, s_h = 5 \cdot$ `sizeof(int)`), while the metadata for `p1` will be ($s_h = (5-2) \cdot$ `sizeof(int)`, $s_l = (5-2-1) \cdot$ `sizeof(int)`). This construction can simplify bounds checking by using only one check instead of two whenever it is possible to infer the "direction" of a memory access: for example, in a common scenario such as a monotonically increasing (or decreasing) loop, the compiler can safely infer the direction and remove one check.

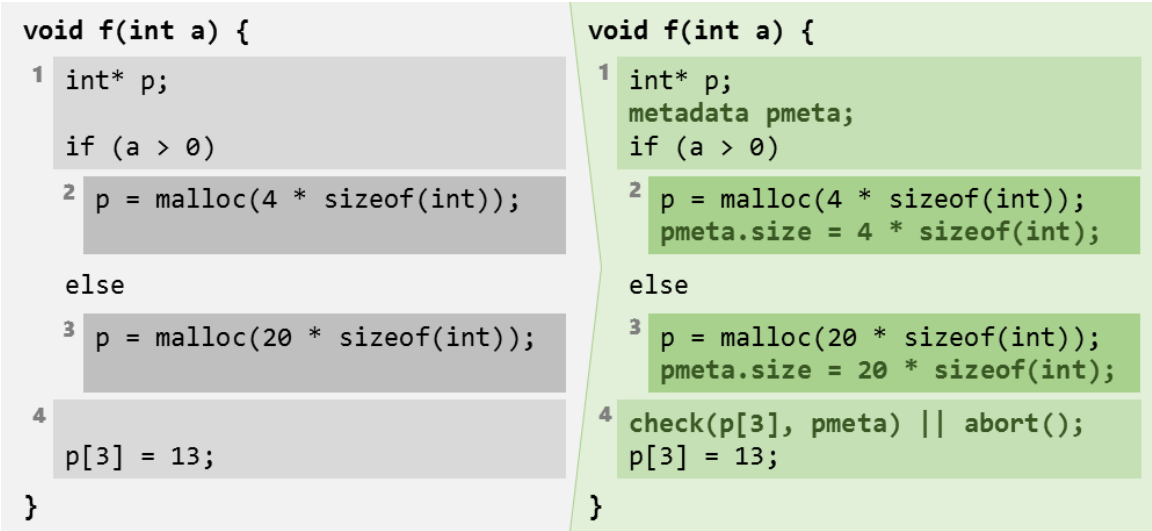


Figure 2.3. Explicit metadata variables.

nesCheck computes the metadata information for each pointer with different strategies, depending on the specific pointer. For static allocations, such as arrays of fixed size or pointers to `structs`, nesCheck directly computes the size of the allocated memory. While dynamic memory allocation is discouraged in TinyOS, nesCheck supports it for completeness. For dynamically allocated memory, the size is computed and updated by keeping track of the parameters of calls to functions such as `malloc()`, `realloc()`, `calloc()`, and `free()`.

In cases where a local pointer can point to different memory areas depending on dynamic control flow conditions, nesCheck generates and injects an explicit variable to hold the metadata for this pointer, depending on the control flow paths. Figure 2.3 shows a concrete example of this scenario – with the original source code on the left and the instrumented one on the right – where function `f()` performs different allocations for pointer `*p` depending on the value of the function parameter `a`. Explicit metadata variables are needed for pointers accessed in basic blocks different than the one they were defined in[1]. In Figure 2.3, the different basic blocks are highlighted as

[1]Detecting this behavior is possible as the heap is in non-SSA form. nesCheck is thus capable of connecting the same logical variable at the different locations (i.e., variable declaration and assignments in disjoint branches).

separate, numbered solid boxes. `*p` is declared in block 1, but is initialized in block 2 or 3, and accessed in block 4.

Metadata Table

A Metadata Table associates specific memory addresses with their metadata information. Efficient data structures, e.g. hashmaps, often use large virtual address spaces [17]. Embedded devices do not have a virtual memory management mechanism, however all the pointers that will need an entry in the Metadata Table are known at compile time, so nesCheck optimizes its data structure by using a dense, array-based Binary Search Tree. Moreover, for code that follows TinyOS's design guidelines and therefore does not make use of dynamic memory allocation, this data structure can be entirely preallocated for a statically-defined size.

We decouple metadata from the pointers – compared to *fat pointers* used in prior work [16,61–65] – in order to achieve a uniform memory representation for all pointers. Moreover, since the search tree is, on average, very small with respect to the total number of pointers, keeping it separate allows nesCheck to choose the optimal data structure.

2.3.2 Dynamic Instrumentation

Dynamic checks can detect all memory errors since they have full runtime view and dynamic information when they are executed. In nesCheck, the metadata for each pointer is set to zero upon declaration, then always kept up-to-date with the actual offsets of the pointer in its memory area. This design allows a single inexpensive bounds check to be effective not only against out-of-bounds memory accesses, but also all other memory errors, such as null/uninitialized pointer dereferences or use-after-free errors.

Every time a dynamic check is necessary, the respective memory access instruction is instrumented to be preceded by a bounds check. A failed check will terminate the

execution and reboot the node, preventing memory corruptions. With no memory separation nor difference between kernel-land and user-land, continuing the software execution after a memory error can have unpredictable, arbitrarily bad outcomes. Rebooting is the only safe fault-handling action to prevent further memory corruption and potential compromising of the entire network on such constrained platforms. The attacker could try to exploit the same vulnerability again, and achieve at best a Denial of Service. Compared to probabilistic defenses, the attacker will never succeed against memory safety. In a debugging scenario, it would be possible to extend our prototype to send an error report message to the base station, including more details about the code location that caused the error. Our current prototype supports the explicit printing of details about the error location on screen when the code is run in a simulator (more about the TinyOS simulator in Section 2.5).

Optimizations

Frequent updates and lookups in the table incur high performance overhead. nesCheck optimizes by adding instrumentation to more directly propagate the metadata.

Functions taking pointers as parameters: A pointer appearing as a parameter in a function will assume different values for different callers of the function. Consequently, the pointer will also inherit different metadata properties depending on the pointer that is passed as actual parameter at every different call site. nesCheck enhances the signature of all the functions that have pointer parameters to include additional parameters for the metadata[2]. As an example, a function with a signature such as `void f(int* p)` is enhanced to `void f(int* p, metadata pmeta)`, where `metadata` is the type of the data structure holding nesCheck's metadata information. Finally, the pointer parameter is associated with the metadata parameter as its own metadata.

[2]Variadic functions are still supported by updates and lookups in the metadata table.

Functions returning pointers: If a function returns a pointer, metadata propagation must also be enabled through the return value. nesCheck enhances the signatures of such functions and their `return` instructions, from a single value to a structure containing the original returned value plus its attached metadata. Thus, the sample function signature `int* f()` will be instrumented into {`int*, metadata`} `f()`. All the return instructions will consequently be transformed from `return p;` into `return {p, pmeta};`, where `pmeta` is the metadata information for pointer `p`. Lastly, all call sites for this function must be instrumented to take into account the change in return type, unpack the two pieces of data from the structure (i.e., the pointer and its metadata) and associate one to the other.

2.3.3 Running Example

In this section, we present the working of the core components of nesCheck on a program example (shown in Figure 2.4) that is small – for ease of detailed discussion and manual analysis of expected behaviors – but stress-intensive in the number of advanced features and memory error corner-cases included. We include casting of pointer types to and from integers, index-based access of memory areas, usage of pointers with incompatible types depending on specified conditions, and dynamic memory allocation as well, even if discouraged by TinyOS, to ensure the correctness of nesCheck even in face of wrong programming styles.

While the analysis and instrumentation of the program in nesCheck is carried out sequentially one entire function at a time, here we follow the execution flow for a more effective presentation. First of all, nesCheck rewrites the signatures for `testMT_aux` to {`foo_t*, meta_t`} `testMT_aux(int* p, meta_t pmeta)`, and instruments in a similar way `testMetadataTable` and `assignLoop`.

In `main()`, nesCheck infers the size for the metadata of `*arr` to be 5 integers, from the parameter of `malloc()`. The subsequent call to `testMetadataTable()` is

```
 1 typedef struct foo {
 2  int  a;
 3  int* bar;
 4 } foo_t;
 5 foo_t myfoo;
 6
 7 foo_t* testMT_aux(int* p) {
 8  foo_t* f = &myfoo;
 9  f->bar = p;
10  return f;
11 }
12 void testMetadataTable(int* p) {
13  foo_t* f = testMT_aux(p);
14  (f->bar)[2] = 13;
15 }
16 void assignLoop(int* p) {
17  int i;
18  for (i = 0; i < 4; i++)
19   *(p + i) = i;
20 }
21 void testDynamicAliasing(int n) {
22  int* p;
23  int a[4];
24  int b[12];
25  if (n < 1) p = a;
26  else       p = b;
27  assignLoop(&(p[1]));
28 }
29 int main() {
30  int* arr = malloc(5 * sizeof(int));
31  testMetadataTable(arr);
32  testDynamicAliasing(0);
33 }
```

Figure 2.4. Representative example for the stress-intensive microbenchmark.

then updated for its new signature (adding as second parameter the metadata for `*arr`), avoiding the need for metadata table accesses.

The `testMetadataTable` function leverages the support function `testMT_aux` for obtaining a pointer to `struct foo_t`, using the characteristic TinyOS pattern of global variables in place of dynamic allocation. The field `f->bar` is aliased to `*p`, and

this time the metadata propagation required metadata table accesses, as the pointer is in a struct. The execution resumes in the `testMetadataTable` function. The storing of a numerical value inside the array member of the `struct foo_t bla` at line 14 is actually translated by Clang into a sequence of `GetElementPtr` statements. Whenever necessary, such instructions are instrumented by dynamic runtime checks and metadata table lookups.

Following the execution, the function `testDynamicallyAliasing`, conceived to stress-test common dynamic aliasing scenarios, is first instrumented with explicit metadata variables, as presented in Section 2.3.1. Then, `assignLoop()` tries to assign numeric values to the first 4 cells of the array, resulting in an out-of-bounds memory violation. However, an injected dynamic runtime check at line 19 will catch the out-of-bounds access to the 4^{th} element of the array, and the execution will be diverted into a trap function.

2.4 Implementation

The implementation of nesCheck leverages the existing TinyOS compiler toolchain and extends it with custom components built on Clang [66] and optimization passes from the LLVM suite [67]. The technologies used are highlighted next to each pipeline block in Figure 2.1.

The nesC source code is initially processed by *ncc*, the nesC compiler, that links the different nesC components together through their interfaces and translates the result to a single C source code file. The C source thes is transformed into the LLVM Intermediate Representation (IR) language. Such IR is a well-specified code representation offering an abstraction layer between the source programming language used (nesC/C) and the actual target platform code. Then, the IR is passed to our nesCheck *Static Analyzer*, based on an LLVM target-independent Optimization Pass.

The nesCheck *Analysis State Manager* component maintains the analysis state throughout the different steps, and propagates information between the various com-

ponents. Most of the metadata is kept in memory by the Analysis State Manager, and looked up and injected only when needed for the appropriate instrumentation.

As a last step, the minimal set of required runtime checks for the memory-manipulating instructions is computed, and the code is instrumented accordingly. The LLVM IR uses, in general, two separate instructions for pointer dereferencing: a `GetElementPtr` instruction to calculate the memory address of the location to be accessed, and a `Load` or `Store` instruction to actually access this memory location and, respectively, place the resulting value in a variable or store a value into the location. nesCheck's instrumentation adds a bounds check conditional branch before the `GetElementPtr` instruction, and a trap function to be invoked whenever the runtime check fails, to terminate the execution and reboot the node, preventing memory corruptions.

Whenever nesCheck can statically determine that any execution of the instruction being instrumented will result in a failure of the check – i.e., the condition can be statically determined to be always false – the user is alerted that a constant memory bug is present, providing her with insights useful to inspect and fix the bug.

The rest of the pipeline, after the instrumentation, resumes the original TinyOS compilation toolchain, having the instrumented code go through the *gcc* compiler to obtain the final native binary for the desired target platform.

2.5 Evaluation

The TinyOS development platform ships with several sample applications, such as radio communication, sensing, and hardware interaction. As done by most other TinyOS research works [18, 68–71], we use these applications as benchmark suite for evaluating nesCheck. Table 2.1 provides details on each program in our benchmark suite. We first use these applications *as-is* to evaluate the performance overhead. Then, we evaluate the overall effectiveness of nesCheck by randomly injecting memory

Table 2.1.
TinyOS standard applications used as benchmark for nesCheck's evaluation.

Application	LOC	Description
BaseStation	5684	Simple Active Message bridge between the serial and radio links.
Blink	5505	Blinks the 3 LEDs on the mote.
MultihopOscilloscope	11728	Data collection: samples default sensor, broadcasts a message every few readings.
Null	4261	An empty skeleton application, useful to test the build environment functionality.
Oscilloscope	6868	Data collection: radio broadcasts a message every 10 readings of default sensor.
Powerup	4306	Turns on red LED on powerup, to test deploy of app on hardware mote.
RadioCountToLeds	6751	Broadcasts a 4Hz counter and displays every received counter on the LEDs.
RadioSenseToLeds	6808	Broadcasts default sensor readings, displays every received counter on the LEDs.
Sense	5699	Periodically samples the default sensor and displays the bottom bits on the LEDs.

bugs in the benchmark applications and verifying that all of them are caught statically or at runtime.

We evaluate nesCheck on several static metrics – such as the number of pointer variables, their inferred type classification, and the number of dynamic check instrumentations – and dynamic metrics – such as the overhead of nesCheck in terms of program size, memory, execution performance, and energy consumption.

To evaluate performance, we compiled the applications for TOSSIM [71], a discrete event simulator, *de facto*-standard tool for TinyOS WSNs. TOSSIM simulates the behavior of TinyOS accurately down to a very low level and precisely times interrupts. This allowed us to perform the evaluation in a controlled environment, through repeatable experiments, and to increase the number of runs for each experiment, while still maintaining a realistic distributed embedded software execution.

Each of the evaluation results has been obtained by averaging 25 independent runs of each test.

2.5.1 Type Inference

The results in Figure 2.5 show that, on average, 81% of the variables are classified as Safe, 13% as Sequence, and 6% as Dynamic. A large number of dynamic runtime checks can thus already be skipped as immediate consequence of the type system inference. Note that, since the analysis is conservative, some pointers classified as dynamic might not be so; however, as shown in the performance evaluation afterwards, this does not degrade the efficiency of our approach.

The average total number of analyzed variables, across all the TinyOS sample applications in the benchmark, is $3,633$, a small number that further supports our design choice of whole-program static analysis.

2.5.2 Code Size and Performance Overhead

We investigate the overhead of nesCheck's instrumentation in terms of code size and performance, and the results are shown in Figure 2.6. The programs in the benchmark total to $57,610$ lines of code. The size overhead is measured in additional bytes the memory-safe executable produced by nesCheck vs. the uninstrumented one, both including the code for the TOSSIM simulator infrastructure. The code size of the uninstrumented programs averages at $228,761$ bytes, and the instrumentation adds only $12,201$ bytes (5.3%) of overhead on average. This result shows that nesCheck is suitable for the instrumentation of programs to be deployed even on devices very constrained in ROM.

We also measure the performance overhead of nesCheck through the TOSSIM simulator for TinyOS. This tool is used by a simulation driver program by repeatedly asking it to execute the next event from the simulation queue. The duration of each event and the total number of events depend on the complexity of the computation

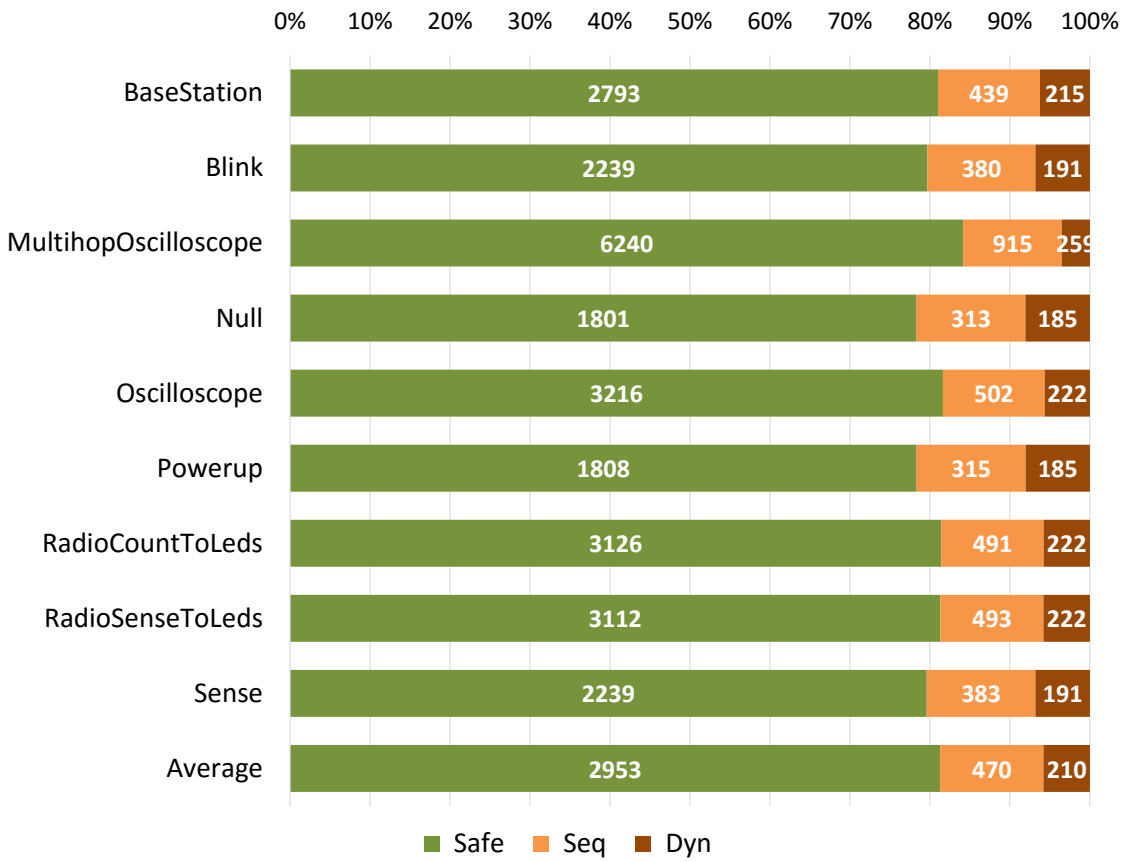


Figure 2.5. Pointer classification results for the TinyOS sample apps benchmark.

to be executed. Therefore, we measure the overhead of nesCheck's instrumentation by fixing the total simulation time to 30 real seconds, running the simulation of the original and instrumented applications, and then measuring the number of simulated seconds actually executed. In three cases (`BaseStation`, `Null` and `PowerUp`), since the applications are merely sample "skeleton" programs to guide developers, no real events were happening after the initial program startup. Therefore, for those programs the reported overhead is 0, and we do not consider them in our averages for the performance overhead. For all the other applications that continuously have processed events, we observe that TOSSIM is able to execute more simulated seconds (in the span of 30 real seconds) for simpler programs than for more complex ones. For example, the simple `Blink` program is executed for 120185.86 simulated seconds,

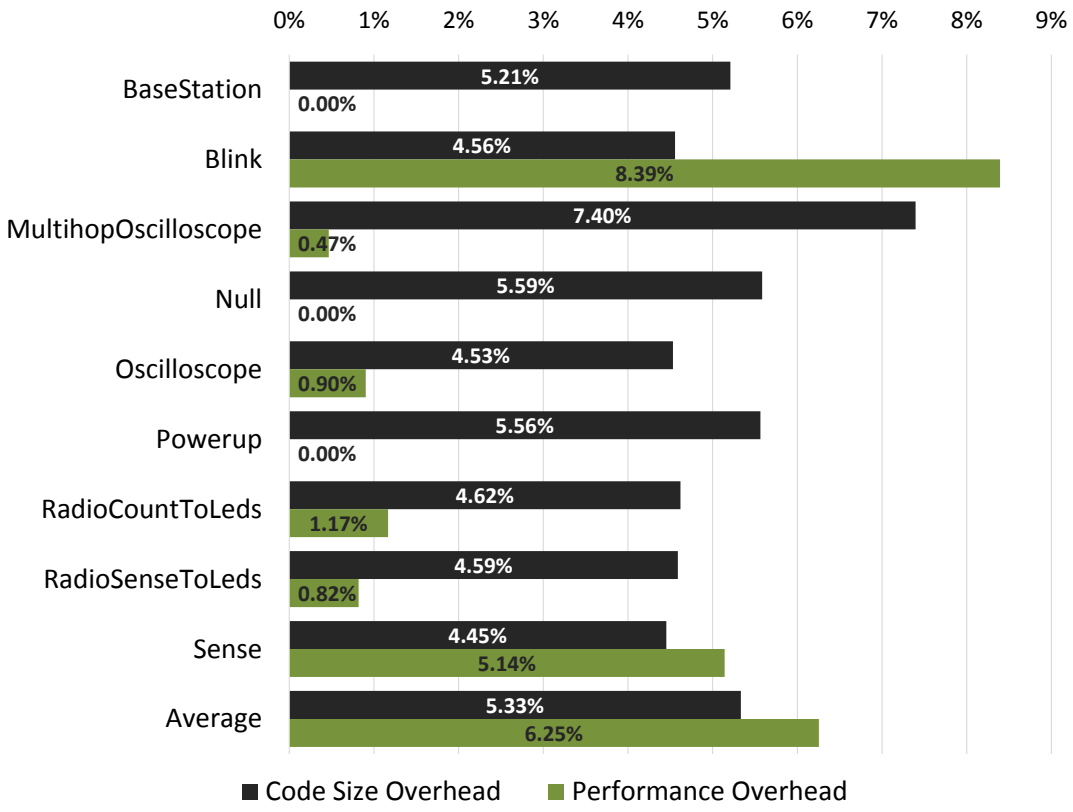


Figure 2.6. Code size and performance overhead for the instrumented TinyOS apps, including TOSSIM.

while the more complex `RadioSenseToLeds` program only reaches 6878.08 simulated seconds (both uninstrumented). In fact, this confirms the intuition that fewer events can be processed in the same time span when the computation of each event is more complex. On average, nesCheck introduces a performance overhead of 6.2%. We note that the maximum overhead (incurred by the `Blink` application) is still quite low, at 8.4%. We believe that this overhead is acceptable for WSN applications.

2.5.3 Memory Overhead

As discussed in Section 2.3.1, some of the pointers require entries in a separated metadata table. We thus measure the impact of this additional data on the memory

of the embedded devices. Figures 2.7 and 2.8 present our results on the memory overhead of nesCheck for the TinyOS applications benchmark. In particular, Figure 2.7 shows the number of metadata lookups added to the code and the number of actual metadata table entries required for each application. On average, nesCheck added only 90 metadata table entry lookup instrumentation points during the instrumentation. Given the SSA form, there is a direct relationship between the number of memory accesses and the number of analyzed variables; therefore, we compare the number of metadata lookups with the total number of variables analyzed by nesCheck, and see that it amounts to just 2%. When only comparing to the Dynamic pointers, it amounts to 41%, which still represents a significant memory saving. Many of such lookups, furthermore, refer to the same logical variable, and thus point to the same entry in the metadata table. Thus, in fact, only 32 distinct entries are needed on average in the metadata table, constituting approximately 1/3 of the total lookup instrumentations for each program.

With these collected metrics, we measure the effective RAM overhead of nesCheck for each application by comparing the RAM occupation of the uninstrumented program – as reported by the nesC toolchain when compiling for the TelosB motes platform [57] – with the size of the metadata table in the instrumented version – representing the effective memory overhead. Figure 2.8 presents both these metrics side by side for ease of presentation. The numbers vary greatly for the different applications, as the number of metadata table entries is completely dependent on the data structures used by each program. However, the average overhead is 16%, and in all cases the total memory requirement remains significantly below the 10kb RAM limit of the TelosB platform chosen for this experiment.

2.5.4 Checks Reduction

As part of our experimental analysis, we collected statistics about the number of runtime checks added to the programs during the instrumentation, together with the

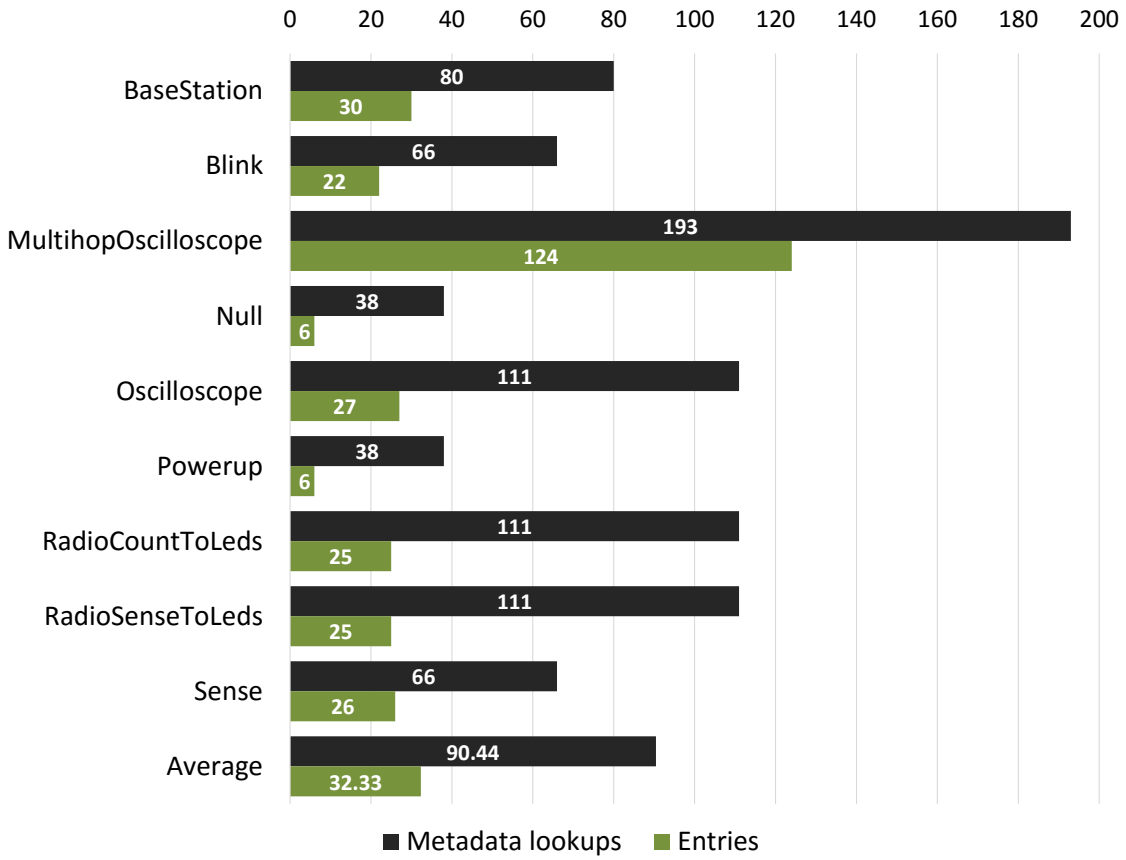


Figure 2.7. Metadata table entry lookups vs. actual metadata table entries required by the instrumentation.

checks that are removed as part of nesCheck's check reduction. As shown in Figure 2.9, the complete analysis and instrumentation of nesCheck for all the TinyOS applications overall reduces, on average, the required checks by 20% of the total potentially vulnerable locations, greatly reducing the performance overhead in enforcing memory safety. For the whole benchmark suite, an average of 452 checks are added, and 110 are skipped.

2.5.5 Energy Overhead

The power consumption for the various operations – such as computation, radio communication, standby or sleep – varies across the different sensor mote hardware

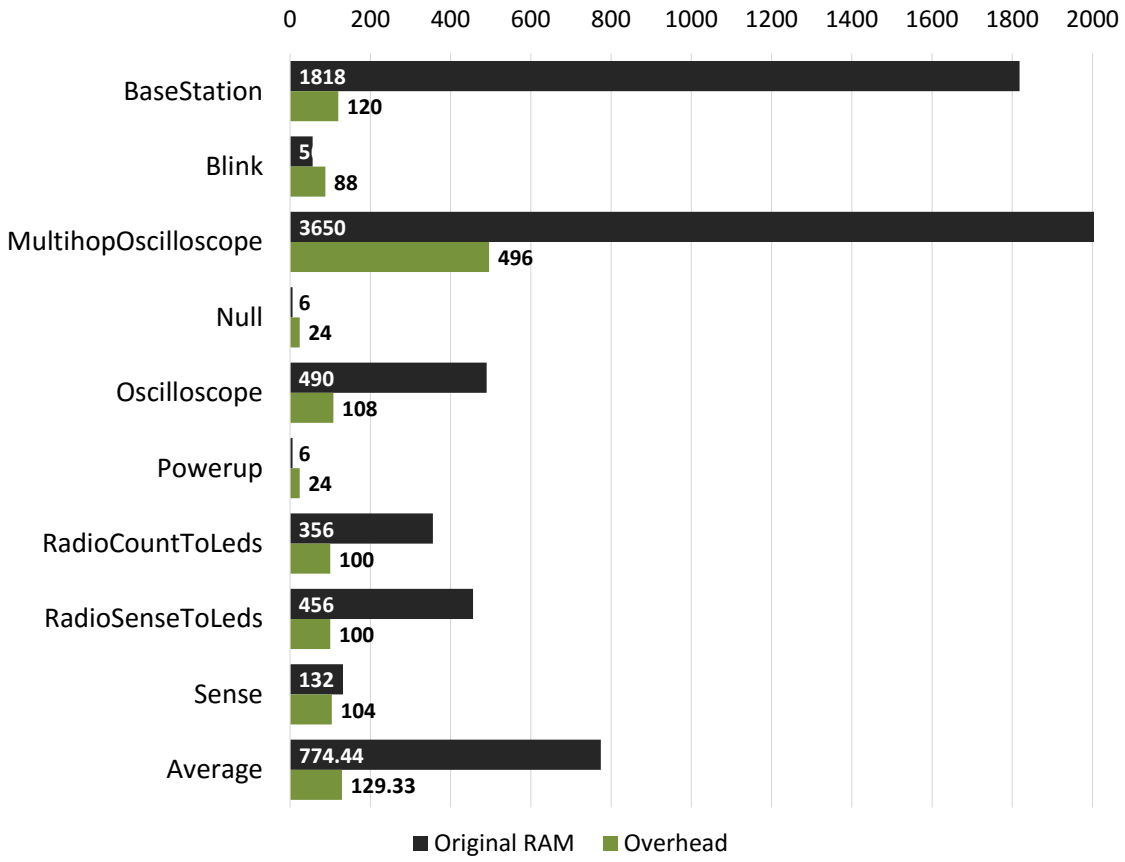


Figure 2.8. RAM occupation of uninstrumented programs and memory overhead of nesCheck (all in bytes).

platforms. However, on all platforms, the majority of the power consumption is always caused by wireless transmission and reception, as well as the transitions between the on and off states of the radio. Shnayder *et al.*, for example, quantitatively measure that, in many cases, active CPU cycles in WSN applications are very small, and have negligible effect on total power consumption [72]. The instrumentation of nesCheck in TinyOS programs does not introduce any additional radio communication, while instead adding some runtime computation for the dynamic checks. Therefore, the energy overhead is, intuitively, proportional to the performance overhead that we measured in our experiments in Section 2.5.2 by a factor of CPU energy consumption.

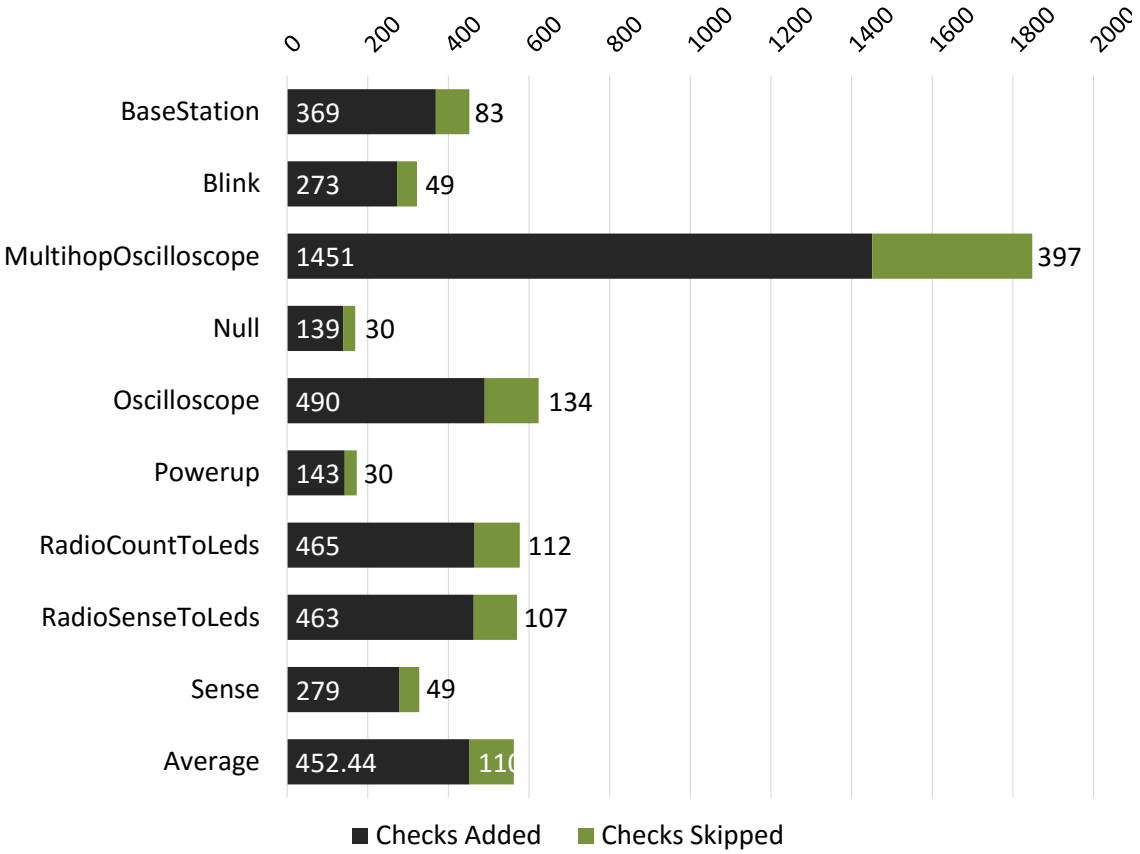


Figure 2.9. Checks added and checks skipped in the instrumented TinyOS sample apps benchmark.

Since measuring the energy consumption directly on the motes' hardware is difficult [73], to quantify this metric we leverage the energy model proposed by Polastre *et al.* [74]. We refer to the MicaZ motes hardware platform datasheet [75] (being the platform simulated by TOSSIM), and multiply the battery voltage by current draw and time. With those calculations, the energy overhead for nesCheck amounts on average to 0.84%, a negligible quantity that supports our analytical expectations.

2.5.6 Fault Injection

To evaluate the effectiveness of nesCheck in preventing all memory errors, we randomly injected memory vulnerabilities and bugs in the TinyOS applications. We

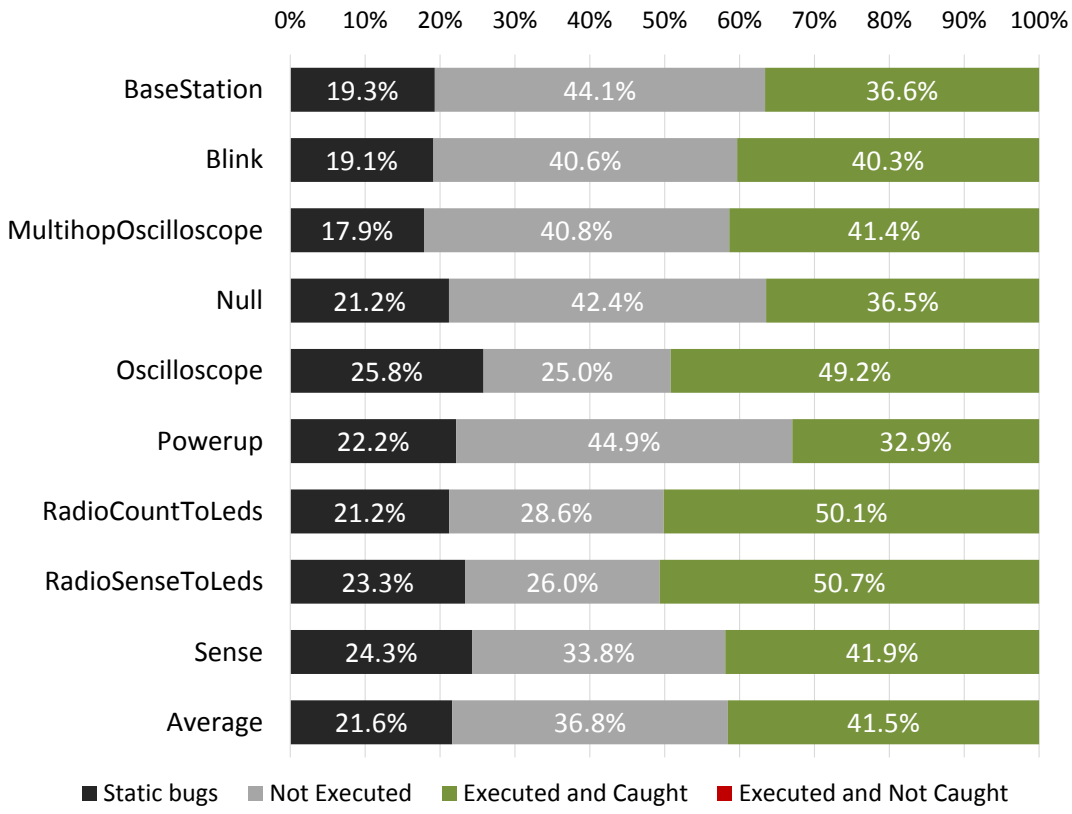


Figure 2.10. Fault injection results on TinyOS benchmark.

injected 500 random faults in each applications in the TinyOS benchmark, for a total of $4,500$ faults. In particular, each time we selected one random memory access, altered its indices to produce a memory error, and included an extra printing instruction to mark the moment when that fault is executed; we then instrumented the application and executed it. We expected nesCheck to correctly find the fault, either statically or at runtime, and prevent the out-of-bounds access.

Figure 2.10 shows our results. On average, 22% of the injected faults were statically caught at compile time. 37% of the faults were injected in areas of the code that were not executed at runtime. For the injected faults that were executed at runtime (41% on average), 100% were correctly caught by the dynamic checks placed by the nesCheck's instrumentation, i.e., no fault was executed and went uncaught.

2.5.7 Naive vs. Optimized Approach

While a direct comparison of nesCheck with traditional techniques such as SoftBound or CCured is infeasible due to (i) constraints of embedded systems, and (ii) the missing implementation of SoftBound or CCured for embedded systems, we measure the performance benefits of nesCheck's check reduction to get an estimate of the improvement over those traditional techniques. We run nesCheck with ("optimized") and without ("naive") check reduction optimizations, and run it on all the applications in the benchmark (excluding those that did not yield events in our performance overhead evaluation in Section 2.5.2). Figure 2.11 shows a comparison of the overhead of the naive and optimized executions of the instrumented programs. We observe an overhead reduction of 41.13% on average, showing how nesCheck's check reduction effectively leads to significant performance improvements.

2.6 Limitations

Currently, a sensor node instrumented with nesCheck is rebooted when a dynamic check fails. Since this might not always be the best, in the future, we plan to work on more advanced, programmer-guided recovery mechanisms, with the goal of maintaining the network as functional as possible even in the presence of memory errors.

More powerful computing platforms (e.g., Raspberry PI) are becoming increasingly available. However, they are impractical for common WSN application purposes, with significantly higher cost, energy requirements, and size, as compared to low-power WSN nodes. The latter have the advantages of being cheap, easily replaceable, deployable in bulk, and in need of little energy. Even when such more advanced devices will become sufficiently cost-effective for large deployments, efficiency would still remain a critical concern for memory safety techniques, as the number and scale of applications deployed on them would consequently increase as well. We plan to work in this direction to investigate how nesCheck can be ported to more power-

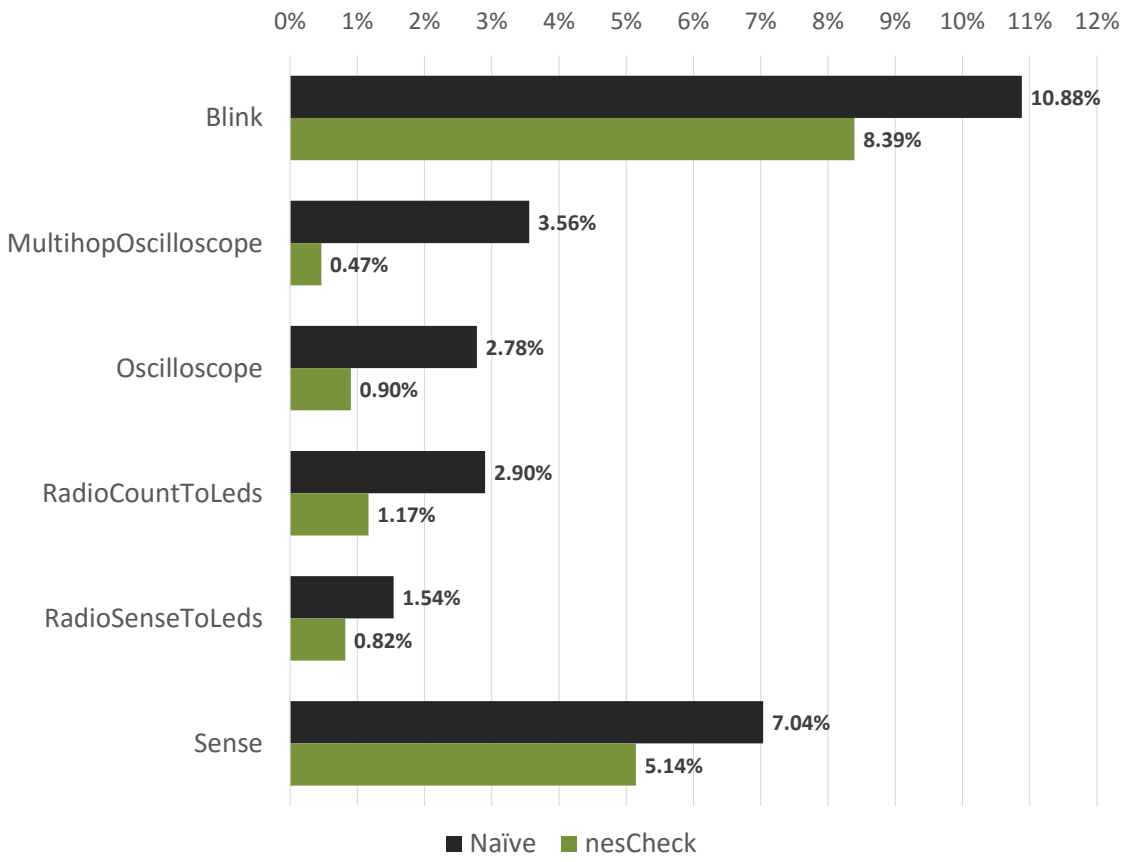


Figure 2.11. Naive vs. optimized instrumentation on TinyOS benchmark.

ful platforms, and leverage the additional capabilities of these platforms to further improve performance.

The current prototype of nesCheck enforces spatial memory safety. Our approach could, however, be extended to also enforce temporal memory safety. Note that most WSN applications do not use dynamic memory allocation, and are therefore fully protected by spatial safety alone. Nevertheless, we plan to extend our implementation of nesCheck to explicitly address temporal safety, and design mechanisms tailored for embedded platforms to enforce it.

Lastly, the scalability of the system, and further overhead reduction, are of great importance. We plan to investigate whether the integration of Bounded Model Checking techniques [76] in nesCheck helps in that direction, as it would enable the use of

formal verification techniques for proving the safety of seemingly dangerous memory accesses, therefore further reducing the overhead. Note that there are several issues that make formal verification on embedded software hard. Several patterns – such as direct communication with hardware registries for sensing, network packets, frequent interrupts, or the use of bit fields – cause the search space for formal verification to quickly explode. Dynamic checks are able to cope with these patterns, at the price of performance.

2.7 Proof of Safety

In this section, we sketch a formal proof of memory safety for nesCheck. First, we give an intuition of the rules for type inference. Then, we follow the general structure of the proof of SoftBound [77], while focusing on the features relevant for nesCheck. We tackle the complexity of the nesC language by focusing the proof on an abstract subset of nesC that captures most of the fundamental primitives. Due to space limitation, we keep the formalism, operational semantics tractation and proof short, while still remaining sound in showing safety.

The syntax we use models programs in their processed IR form, already reduced to atomic data types (`int` and pointers) and simple operations. Table 2.2 shows the grammar we consider for our proof. We use *RHS* and *LHS* to denote left-hand side and right-hand side, respectively. Note that, while most WSN applications do not use dynamic memory allocation, we include it in our formal grammar for the sake of generality. In our simplified operational semantics, we consider an environment E that models the stack with a map S from variable names to addresses and types, models the type inference with a map Γ from variable names to pointer categories, and models the heap with a partial map M from addresses to values.

Using `some` and `none` to denote presence or absence of a value, we model nesC's memory access primitives as follows: (i) `read` M l: if l is an allocated memory location, return `some`, otherwise return `none`; (ii) `write` M l v: if l is an allocated

Table 2.2.
Grammar used in the formal proof of safety.

Atomic Types	t	::=	`int` | p*
Pointer Types	p	::=	t | s | `void`
Struct Types	s	::=	`struct`{f ; f}
Struct Fields	f	::=	(id:a)
LHS Expressions	lhs	::=	x | *lhs | lhs.id | lhs->id
RHS Expressions	rhs	::=	val | rhs+rhs | lhs | &lhs
		|	(`(`a`)`rhs | `sizeof(`p`)`
		|	`malloc(`rhs`)`
Commands	c	::=	c ; c | lhs = rhs

memory location, set the content to the value v; (iii) `malloc` M s: if M has an available region of size s, allocate and return it, otherwise fail.

The normal C operational semantics processes assignments by writing the result of the expression in the RHS operand to the address calculated from the LHS operand. For this proof, we extend the traditional operational semantics of C by including new outcomes for operations (that include memory errors) and tracking of pointers metadata. A result r can therefore be: (i) $v_{(sl,sh)}$, a value v with the attached metadata for the size of the memory region towards the lower (sl) and higher (sh) memory addresses (see Section 2.3.1); (ii) a memory address l; (iii) `Success`; (iv) `MemoryError` if a bounds check failed; (v) `MemoryExhaustion` if M did not have enough free memory upon a `malloc` operation.

Using the above definitions, we formalize nesCheck's operational semantics with four classes of rules. First, the rules for type inference and propagation. Second, the $(E, lhs) \Rightarrow_l r : a$ rule specifies how LHS expressions are evaluated (no changes to the environment). Third, the $(E, rhs) \Rightarrow_r (E', r : a)$ rule specifies how RHS expressions are evaluated (potential changes to the environment; if successful, r is $v_{(sl,sh)}$). Lastly, $(E, c) \Rightarrow_c (E', r : a)$ is the rule to execute commands (r must be a success or failure result). Here we omit rules straightforwardly representing standard C semantics, and just show the rules most relevant for nesCheck's semantics.

Type Inference. We present some of the rules for type inference, that formalize the rules presented in Section 2.3.1. For example, pointer arithmetic on a Safe or Sequence pointer causes the result to be of Sequence kind, while casting a Safe or Sequence pointer to an incompatible type[3] results in a Dynamic pointer:

$$\text{Types}\frac{\Gamma(x) = \tau \quad \tau \in \{\texttt{Safe}, \texttt{Seq}, \texttt{Dyn}\}}{\Gamma \vdash x : \tau}$$

$$\text{ArithT1}\frac{\Gamma \vdash e_1 : \tau \quad \tau \in \{\texttt{Safe}, \texttt{Seq}\} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{Seq}} \qquad \text{ArithT2}\frac{\Gamma \vdash e_1 : \tau \quad \tau = \texttt{Dyn} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{Dyn}}$$

$$\text{IllegCast}\frac{(E, x) \Rightarrow_l l : t \quad \mathit{incompatible}(t, t')}{\Gamma \vdash (t')x : \texttt{Dyn}}$$

No memory access to Safe pointers is subject to dynamic bounds checks; conversely, all memory accesses to Dynamic pointers are instrumented with runtime checks. For a memory access to a Sequence pointer, if nesCheck can statically determine that it will never result in an out of bounds operation, it will not be instrumented with a dynamic check. We therefore define a predicate $safe(*p)$ that is true ($\top$) iff the memory access $*p$ does not require bounds checks, false ($\bot$) otherwise:

$$safe(*p) = \begin{cases} \top & \text{if } \Gamma \vdash p : \texttt{Safe} \vee \\ & \quad (\Gamma \vdash p : \texttt{Seq} \wedge *p \text{ not out of bounds}) \\ \bot & \text{otherwise} \end{cases}$$

nesCheck declares a memory access as never out of bounds only if the in-memory metadata propagated up to that access statically indicates the safety of the operation. The formal proof of CCured [16] shows it is safe to leave memory accesses uninstrumented, and the same proof also applies in our case to statically-provable Sequence pointers accesses.

[3] In Section 2.3.1, we define two types as "incompatible" when, for example, they have different levels of indirection or have same level of indirection but different root types.

Dynamic Bounds Checks. The bounds checking operational rules are very similar to those of SoftBound, so we will omit most of them for brevity. We present here the rules for the evaluation of a pointer dereference operation, both in case of success:

$$\text{DerefSuccess}\frac{\begin{array}{c}(E, lhs) \Rightarrow_l l : t* \\ \texttt{read}\ (E.M)\ l = \texttt{some}\ v_{(sl,sh)} \\ safe(l) \vee (sl \geq 0 \wedge sh \geq sizeof(t))\end{array}}{(E, *lhs) \Rightarrow_l v : t}$$

and in the case of memory error (failed bounds check):

$$\text{DerefFail}\frac{\begin{array}{c}(E, lhs) \Rightarrow_l l : t* \\ \texttt{read}\ (E.M)\ l = \texttt{some}\ v_{(sl,sh)} \\ sl < 0 \vee sh < sizeof(t)\end{array}}{(E, *lhs) \Rightarrow_l \texttt{MemoryError} : t}$$

Other rules, such as those for type casts and pointer arithmetic, need to ensure that metadata information is propagated correctly:

$$\text{PtrArithm}\frac{\begin{array}{c}(E, ptr) \Rightarrow_r (E', l_{(sl,sh)} : p*) \\ (E', val) \Rightarrow_r (E'', off_{(sl',sh')} : \texttt{int}) \\ l' = l + off * sizeof(p) \\ sl' = sl + off * sizeof(p) \\ sh' = sh - off * sizeof(p)\end{array}}{(E, ptr + val) \Rightarrow_r (E', l'_{(sl',sh')} : p*)}$$

$$\text{TypeCast}\frac{\begin{array}{c}(E, rhs) \Rightarrow_r (E', v_{(sl,sh)} : t) \\ t' \neq \texttt{int}\end{array}}{(E, (t')rhs) \Rightarrow_r (E', v_{(sl,sh)} : t')}$$

For the formal rule for integer-to-pointer cast, we follow SoftBound's approach of zeroing out the metadata to avoid potentially undefined behaviors:

$$\text{TypeCastIntToPtr}\frac{\begin{array}{c}(E, rhs) \Rightarrow_r (E', v_{(sl,sh)} : t) \\ t = \texttt{int} \\ (sl', sh') = (0, 0)\end{array}}{(E, (t')rhs) \Rightarrow_r (E', v_{(sl',sh')} : t')}$$

With this support infrastructure of rules in place, we note that the operational rules for values that are valid at runtime and need runtime bounds checks are fully equivalent to their corresponding rules in SoftBound's formal model [17]. Therefore,

they satisfy the same safety invariants and ensure memory safety for those values, as proven for SoftBound (in Theorems 4.1 and 4.2, and Corollary 4.1 in [17]).

While adding bounds checks to every memory access is surely sound, as shown by the proof in SoftBound, by combining the latter with the proof in CCured we improve the performance overhead by removing unnecessary checks while still remaining sound. Thus, given the operational semantics rules above, every memory access in nesCheck is either safe at runtime – resulting in a correct access – or causes the application to stop – due to a detected memory error. Therefore, the nesC applications analyzed and instrumented by nesCheck fulfill the set memory safety goals.

2.8 Related Work

Memory safety is an ongoing research topic [78]. Attacks to WSN software through memory vulnerabilities have been widely investigated. Against common belief that Harvard-architecture devices would prevent code injection attacks, Francillon *et al.* [50] showed a detailed exploit for code injection without size limitation through carefully crafted network packets. Giannetsos *et al.* carried out a similar study [51], targeting Von Neumann-architecture devices. These two works cover most common architectures for WSNs, able to exploit, for example, both MicaZ and TelosB motes.

From the defense point of view, research work has typically taken three different directions: runtime protection, formal analysis and symbolic execution. nesCheck uses an approach that enhances the runtime protection class of mechanisms with static analysis techniques. Note that all related work mostly focuses on either spatial or temporal safety, since addressing both in a same technique is too complex.

Runtime protection. Necula *et al.* introduce an extended type system for CCured [16]. CCured uses pointer classification as a static analysis technique to infer safe pointers that do not need bounds checks; however, it instruments all non-safe pointers in the code with runtime checks, potentially generating many unnecessary checks. nesCheck overcomes this issue by leveraging more extensive static analysis

techniques to conservatively detect whether some of the sequence pointers can be left unchecked too, as well as detecting statically-recognizable memory violations.

SoftBound [17] is a compile-time approach that instruments C code to enforce spatial memory safety by (i) keeping track of the properties of each the pointed memory area, and (ii) wrapping each memory access instruction with a bounds check. This approach was designed for desktop system and not embedded devices. While nesCheck too leverages dynamic runtime checking to enforce memory safety, it tailors and optimizes this approach to the specific characteristics of nesC applications in order to improve performance.

Compared to the notable solutions just discussed, as well as other traditional ones, nesCheck works for embedded software, being designed specifically for their constraints, challenges, and advantages.

One of the most relevant approaches for memory protection in WSN applications – and TinyOS in particular – is Safe TinyOS [18]. Cooprider *et al.* investigate issues related to the implementation of memory protection for TinyOS programs by formalizing the problem and the requirements, and developing optimizations that make runtime checks more viable under the strict performance constraints of WSN software. Safe TinyOS relies on the Deputy source-to-source compiler [79] to infer necessary information for the code instrumentation[4]. Safe TinyOS, however, puts much of the analysis burden on the programmers, requiring them to either annotate the code with specific type definitions and safety guidelines, or to declare entire components as "trusted" and therefore skipped by the tool. nesCheck, on the other hand, automates the entire process, with no need for source code modifications. Also, nesCheck reduces the potential runtime overhead by removing unnecessary checks before the instrumentation.

Formal analysis. Bucur *et al.* [68] propose a source-to-source transformation tool to make TinyOS code processable by the CBMC [80] bounded model checking [76] proving tool. The well-known limitations of formal verification, in particular the

[4]The Deputy project, however, is no longer maintained.

search space explosion, are inherited by this approach too. Even though Bucur *et al.* propose several optimizations to reduce the complexity to be handled, large-scale applications can still suffer by long times for analysis and potential undecidability if the state becomes too big to be handled.

Symbolic execution. Sasnauskas *et al.* [70] build an approach on top of the Klee symbolic execution framework to debug TinyOS applications before deployment. Just like for the formal analysis-based approaches, the bottlenecks for these designs are: (i) the need for a good model definition of the application to be tested, and (ii) the rapid explosion of the search state. If either part of the design results in a non-complete coverage of every possible vulnerability, then not all the bugs can be effectively identified. Conversely, since nesCheck leverages runtime checks for all the memory accesses that cannot be statically proven as safe, in a conservative way, nesCheck is guaranteed to always catch all the potential vulnerabilities and prevent memory corruption.

Hardware. Not belonging to any of the three categories of defense research directions, Francillon *et al.* [52] propose a hardware modification to split the stack in a control flow stack and a data stack. While this is an interesting idea, it would require hardware manufacturers to change the platform (an economically burdensome path unlikely to be pursuable). nesCheck's software-only approach does not require changes to the hardware platform, nor to the source code.

2.9 Summary

This chapter presented nesCheck, a novel approach that combines whole-program static analysis and dynamic checking techniques to efficiently enforce memory safety on nesC programs, without requiring any source modification. Among the contributions of this work, we design an inter-procedural whole-program static analysis mechanism – based on type tracking and pointer usage and without the need for programmer annotations – and a dynamic instrumentation technique for efficient mem-

ory safety enforcement on highly constrained embedded platforms, without MMU or kernel/user space separation.

Even with efficient techniques, memory protection comes with a performance cost. Therefore, it is important to strategically plan which devices it is more critical to protect, as well as allocate any additional security resource accordingly. We address this problem in the work presented in the next chapter.

3 STRATEGIC ALLOCATION OF SECURITY RESOURCES FOR IOT

The technique presented in the previous chapter provided hardening to the individual sensor nodes from the point of view of memory safety. It is however critical that an effective security infrastructure be adopted throughout the whole network, considering the available security resources and existing protection level of each node. An important requirement for such an infrastructure is to carefully address resource efficiency, due to constrained resources. Its cost would in fact outweigh the low cost of IoT technologies. From these observations it follows that, besides effectiveness, efficiency is a main goal driving the design of an IoT security infrastructure. Efficiency, as well as effectiveness, depends on the choice of the security resources (IDS, special hardware, additional devices, etc), and how they are allocated in the system of interest. Among all possible resource allocation plans, some are more efficient – as they require a lower amount of energy consumption and/or entail a cheaper cost in terms of additional equipment, while other plans are more effective – because harder to be evaded by the attackers. In the work presented in this chapter, we focus on the problem of effectively and efficiently securing IoT networks, and propose a method to compute a resource allocation plan as a Pareto-optimal solution. We start from the assumption that, for each available security resource, we can estimate its installation cost and its average energy consumption at operating speed. We can thus estimate the efficiency of an allocation plan, based on the total energy consumption of the security infrastructure and the costs of its components. To measure the effectiveness of an allocation plan we use two metrics, namely *risk* and *criticality*. The former is defined as the maximum number of network nodes that are no longer protected when an attacker succeeds in taking down at least one security resource. The latter is a measure of how critical certain nodes are for the correct operation of the network. We identify a set of heuristics for computing the criticality value of each

node of the IoT network of interest. We model the interaction between attacker and defender as a Stackelberg leadership model in which the leader (the defender) moves before the follower (the attacker) [81]. We show how to compute a Pareto-optimal defender strategy (resource allocation plan) in two steps. First, among all possible allocation plans, we compute the subset of the Pareto-optimal plans, as the solutions of a three-objective optimization problem, which minimize *(i)* the total energy consumption, *(ii)* the installation cost, and *(iii)* the maximum criticality. In the second step, we select the plan, among those identified in the previous step, which minimizes the risk – as the solution of a single-objective optimization problem. The resulting defender strategy is efficiency-optimal because entails the lowest energy consumption and the cheapest installation cost, while its effectiveness lies in the fact that the number of security resources an attacker needs to take down is maximized – and consequently, the probability of a successful attack is reduced. We have implemented our algorithms and tested them with different network topologies. Our results show that the proposed algorithm provides security infrastructure options for IoT networks that represent different combinations of energy consumption, cost, and probability of successful attacks.

The framework provides the following main features:

- the formalization of a process that is critical and often done in an ad-hoc manner by administrators;
- a method for computing the best defender strategy, that gives to security managers the possibility to choose the resource allocation plan that best fits their efficiency and effectiveness requirements;
- the formulation of the defender's problem as a linear optimization problem;
- a heuristic for the formalization of the equations that define the IoT scenario and the security goals;
- measures to formalize efficiency and effectiveness of the defender strategy.

3.1 Threat Model

While our approach is independent from the actual attack, we assume that the attacker can take any of the steps commonly used to carry out attacks, such as physically tampering with the network nodes, capturing and reprogramming legitimate nodes and security resources, and adding malicious entities to the network to overhear data communications, inject false data and control traffic, intercept and drop data packets, introduce interference, claim multiple identities, and more. We further assume that the attacker must compromise at least one security resource to carry out an attack, is aware of the defense strategy, and smartly targets the most critical security resource.

3.2 Security Model and Definitions

The goal of the defender is to secure the network. To do so, the defender needs to choose a set of security resources among the available ones, and decide where to deploy them in the network area for securing all the network nodes. Due to the heterogeneity of IoT systems, there are different kinds of security resources the defender may have to use. Intrusion detection systems [26] as well as attack prevention systems [82] can be installed on network nodes to, respectively, detect ongoing, and prevent future attacks; physical tools, like directional antennas and highly sensitive transceivers [83, 84] can be installed on watchdogs nodes, thus enabling these nodes to better control other nodes [85]; nodes with tamper resistant hardware can replace normal network nodes in order to strengthen the network [86]; nodes with specialized hardware can be used to implement specific security techniques [87, 88]; additional nodes can be deployed at specific locations in the network with the sole purpose of executing security tasks [85].

3.2.1 Basic Concepts and Notation

Here we introduce the basic concepts and notations that help us to formalize the IoT environment and the security requisites for the defender.

Definition 3.2.1 (security resource) *A* security resource *is a tuple* $sr = \langle c, e, t, loc \rangle$*, where:*

- c *is the cost;*
- e *is the energy consumption;*
- t *is the resource's type;*
- loc *is the location.*[1]

Definition 3.2.2 (network) *A* network *is an undirected graph* $N = \langle V, E \rangle$*, where:*

- $V = \{(e, loc, cr) : e$ *is the node energy consumption,* loc *is the node location, and* cr *is the criticality value*$\}$ *is the set of network nodes;*
- $E = \{\{v_1, v_2\} : v_1$ *is in the communication range of* v_2*, and vice versa, and* $v_1, v_2 \in V\}$ *is the set of links.*

We define a network as an undirected graph given the *physical*-based topology (inferred by the communication range of network nodes), different from the *routing*-based topology (always a sub-graph of the physical one), regardless of routing protocol and direction of the links. The notation $v.x$ and $sr.y$ denotes attributes x of node v and y of resource sr. Given a network N and a set of security resources $\mathcal{R}$, we define:

- a resource allocation plan AP for N as a subset of $\mathcal{R}$, $AP \subseteq \mathcal{R}$;
- $\mathcal{AP}$ as the set of all possible resource allocation plans for N;

[1]The attribute *loc* in Definition 3.2.1 helps us to simplify the formalization of the linear programs we show hereafter. The basic idea is the following: given that in the network area there are many locations where a security resource can be placed, for each resource sr^* we assume to have $sr_1, \ldots sr_n$ resources, one for each location where sr^* can be located.

- the *criticality* of a network node $n \in N$ as the measure of its relevance for the correct function of the network;

- function $crit : \mathcal{R} \times \mathcal{AP} \times N \rightarrow \mathbb{N}$ as the criticality associated with a security resource sr; given an allocation plan $AP \in \mathcal{AP}$ and a network N, $crit(sr, AP, N)$ returns the maximum criticality value over the set of nodes of N that are not any longer protected if sr stops working and no other security resource $sr' \in AP$ covers them;

- function $risk : \mathcal{R} \times \mathcal{AP} \times N \rightarrow \mathbb{N}$ as the risk associated with a resource sr; given an allocation plan $AP \in \mathcal{AP}$ and a network N, $risk(sr, AP, N)$ returns the number of nodes of N that are not any longer protected if sr stops working and no other security resource $sr' \in AP$ covers them.

- the function $Neighbors : V \rightarrow 2^V$,
  such that $Neighbors(v) = \{v' : \{v, v'\} \in E\}$;

- the function $Edges : V \rightarrow 2^E$,
  such that $Edges(v) = \{\{v, v'\} : v' \in Neighbors(v)\}$;

- the set of locations as $\mathcal{L} = L_V \cup L_A$, where L_V is the set of locations taken by each $v \in V$, and L_A is the set of all other available locations in the network area;

- the function $Res : 2^{\mathcal{L}} \rightarrow 2^{\mathcal{R}}$, such that, given a set of locations X, $Res(X) = \{sr : sr.loc \in X\}$, i.e., the resources that can be placed on locations of X;

- the *domain* of a security resource sr as the set of locations that are in the action range of sr, i.e., the portion of network area covered by sr;

- the function $Dom : \mathcal{R} \rightarrow 2^{\mathcal{L}}$, such that $Dom(sr)$ returns the set of locations that belong to the domain of sr;

- $T = \{sr.t : sr \in \mathcal{R}\}$ as the set of security resources types.

Example 1 *Figure 3.1 shows a network with 7 nodes protected by two security resources* sr_1 *and* sr_2*. The main square which contains the network represents the network area. Security resources can be deployed in the locations of the network area, represented by internals dotted squares. The dashed circles denote the action range of the security resources. The number associated to each node denotes both the criticality value and the node id. If one of the two security resources stops working, the set of nodes left without protection is* $S_1 = \{1, 4, 5\}$ *for* sr_1*, and* $S_2 = \{2, 3, 6\}$ *for* sr_2*. The node with maximum criticality in* S_1 *is node 5, while in* S_2 *is node 6. The criticality and risk values of the two security resources, thus, are:* $crit(sr_1, AP, N) = 5$*,* $crit(sr_2, AP, N) = 6$*, and* $risk(sr_1, AP, N) = risk(sr_2, AP, N) = |S_1 \models |S_2 \models 3$*, with* $AP = \{sr_1, sr_2\}$*. Furthermore, we have that:*

- $Neighbors(1) = \{4, 7\}$*,* $Neighbors(2) = \{3, 7\}$*, ...;*
- $Edges(1)$=$\{\{1, 4\}, \{1, 7\}\}$*,* $Edges(2)$=$\{\{2, 7\}, \{2, 3\}\}$*, ...;*
- $\mathcal{L} = \{\langle x_1, x_2\rangle : x_1, x_2 \in \{A, B, C, D, E, F, G\}\}$*;*
- $L_V = \{\langle B, B\rangle, \langle B, D\rangle, \langle D, B\rangle, \langle D, D\rangle, \langle D, F\rangle, \ldots\}$*;*
- $L_A = \mathcal{L} \setminus L_V$*;*
- $Res(\langle C, C\rangle) = \{sr_1\}$*,* $Res(\langle E, E\rangle) = \{sr_2\}$*,*
  $Res(\langle C, C\rangle, \langle E, E\rangle) = \{sr_1, sr_2\}$*;*
- $Dom(sr_1) = \{\langle x_1, x_2\rangle : x_1, x_2 \in \{B, C, D\}\}$*,*
  $Dom(sr_2) = \{\langle x_1, x_2\rangle : x_1, x_2 \in \{D, E, F\}\}$*.*

The idea behind the notion of risk lies in the fact that the minimization of the maximum risk restricts the operating range of the attacker, in case (s)he manages to compromise a security resource. Thus, (s)he probably will need to compromise more than one resource in order to carry out the attack. For instance, during a *sybil attack* [89] on a wireless sensor network, a malicious node presents multiple identities

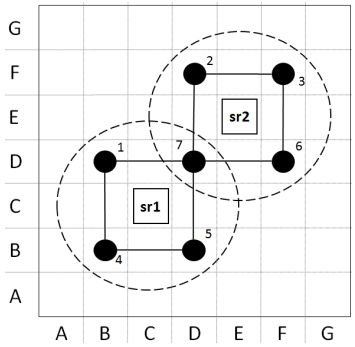


Figure 3.1. An example network.

to other nodes. The attack is much more effective when those identities belong to real nodes of the network. Thus, the attacker needs to compromise a certain number of nodes, from which to steal the identity, before the one from which to start the attack. However, if the defender strategy is well designed, the attacker will probably have to damage more than one security resource in order to steal a sufficient number of identities, therefore delaying the attack and increasing the risk of being detected. Other types of attacks, instead, need just to compromise one node. This is the case of the *black hole* attack [89], in which the compromised node drops all the incoming packets. A black hole is much more effective when the attacker chooses to compromise a critical node, such as one with a high incoming traffic rate. In this case, a defender strategy must provide stronger coverage for the most critical nodes, i.e., those that are more relevant for the correct function of the network. An attacker will thus be forced to compromise more than one resource in order to compromise the most critical nodes.

3.2.2 Definition of Secure Network

A formal definition of *secure network* that matches the techniques adopted by the security systems is crucial, since it gives the guidelines for the formalization of the linear constraints we will use in the Pareto analysis. We can classify security systems

into two main categories: *detection* and *prevention* systems. Those categories correspond to two different security policies: *(i) node/link monitoring*, and *(ii) node/link hardening*. As an example of link monitoring we mention the IoT IDS by Raza *et al.* [26], which checks the link quality for the detection of network layer and routing attacks, and the Liteworp system [90] that uses guard nodes as a countermeasure for wormhole attacks in WSNs. We would like also to mention Dataguard [91] as an example of node monitoring based on a code attestation technique to check the presence of malicious code in the memory of network nodes. Titan [86] is an example of a security architecture for hardening tiny devices with a hardware-assisted dynamic root of trust. The encryption key management scheme by Eschenauer *et al.* [92], to harden links among nodes neighbors, is an example of link hardening. According to those two security policies, we derive the definition of secure network.

Definition 3.2.3 (secure link) *A link* {v,v'} *is* secure *if at least one of the following conditions holds:*

(σ) *both nodes* v *and* v' *are in the communication range of the same watchdog;*

(γ) *both nodes* v *and* v' *can establish a secure communication channel.*

Definition 3.2.4 (secure node) *A node* v *is* secure *if at least one of the following conditions holds:*

(α) v *is tamper resistant;*

(β) *every link that involves* v *is secure.*

In condition (α) the term *tamper resistant* means that the node is equipped with a security tool that makes it inaccessible to attackers wishing to compromise it. In other words, Definition 3.2.4 states that the correct functioning of a node can be guaranteed by avoiding malicious code injection, or by monitoring/strengthening the links connecting to the other nodes. In Definition 3.2.3, a *watchdog* [29] is a security resource that monitors network nodes behavior. Past approaches to intrusion detection

use watchdogs for overhearing in/out-coming traffic of neighbors nodes, performing code attestation, checking the signal strength, etc. Conditions (α) and (γ) capture the policy adopted by prevention systems (node/link hardening), while conditions (β) and (σ) the policy adopted by detection systems (node/link monitoring).

We can now define a network $N = \langle V, E \rangle$ as secure if each node $v \in V$ is secure, according to Definition 3.2.4.

3.3 Players' Strategy

In this section, we define the concept of "strategy" for the defender and the attacker. Their interaction is modeled as a Stackelberg game [81]. In such a game the defender plays the leader and makes the first move by installing a security infrastructure AP. The attacker plays the follower by trying to compromise one or more security resources $sr \in AP$, so that the attack can be carried out on the nodes that are no longer protected by the damaged security resources.

3.3.1 Defender's Strategy

The defender strategy consists of a security resource allocation plan. Given that each security resource entails an installation cost and some energy consumption, the best resource allocation plan for the defender is the one that provides a reasonable balance between efficiency, in terms of energy consumption and cost, and effectiveness, in terms of maximum risk and maximum criticality. The best plan is computed in two steps. In the first step we perform a Pareto analysis which solves the optimization problem defined by the following equation:

$$\min_{AP \in \mathcal{AP}} \{ec(AP), tc(AP), \max_{sr \in AP} crit(sr, AP, N)\} \tag{3.1}$$

where $ec(AP)$ and $tc(AP)$ denote the total energy consumption and the total cost of the allocation plan AP, respectively, and $\max_{sr \in AP} crit(sr, AP, N)$ is the maximum

criticality value over the set of nodes of N that are no longer protected when one resource in AP stops working. The Pareto analysis consists in computing a set of Pareto points $p = (ec, tc, cr)$, that we refer to as Pareto curve. Each point corresponds to a set of allocation plans, i.e., all the plans (strategies) that have an energy consumption, a total cost, and a maximum criticality equal to the values of ec, tc and cr, respectively. Then, we choose the point p^+ whose values best fit our efficiency and criticality requirements. In the second step we compute the best allocation plan (best defender strategy), by solving the optimization problem defined by the following equation:

$$\min_{AP \in AP^+} \{ \max_{sr \in AP} risk(sr, AP, N) \} \tag{3.2}$$

where $AP^+ \subseteq \mathcal{AP}$ is the set of allocation plans that entail an energy consumption, a cost, and a maximum criticality as the values of the Pareto point p^+ chosen in the previous step; and $\max_{sr \in AP} risk(sr, AP, N)$ is the maximum number of nodes of N that remain unprotected when a security resource in AP stops working.

3.3.2 Attacker's Strategy

We assume that an attacker needs to compromise at least one security resource in order to carry out an attack. In fact, whatever is the attack, an attacker always needs to first open a breach in the security, before attacking the system. We address the worst case attacker, that is the one who knows the defenders strategy AP, and plays her/his best strategy, i.e., chooses to compromise the security resource that maximizes the risk (i.e., leaves unprotected the maximum number of nodes), the criticality (i.e., leaves unprotected the most critical nodes), or a combination of both. An attacker strategy is represented by a security resource $sr \in AP$. The best attacker strategy is defined as follows:

$$sr^* = \max_{sr \in AP} \alpha \cdot risk(sr, AP, N) + \beta \cdot crit(sr, AP, N), \alpha + \beta = 1 \tag{3.3}$$

3.4 Computing the Defender's Strategy

In this section we first introduce a heuristic for computing node's criticality, then we briefly survey the concept of Pareto analysis. Finally we describe the two steps for computing the optimal defender strategy.

3.4.1 Computing Node's Criticality

Computing the criticality of each node is a key aspect of our approach. The more accurately we identify the "relevance" of each node for the correct function of the network, the better the defender strategy works. Due to the heterogeneous nature of IoT devices and IoT networks, it is not possible to design a general method for computing nodes criticality which work well for all IoT scenarios. Rather, we encourage network administrators to follow an ad-hoc approach for each specific case. However, we propose a two-steps approach to determine criticality, that we believe can be adopted for the majority of the existent IoT scenarios. The first step assigns a criticality value to each node based on the rate of their in/out-going traffic. Nodes that are more fundamental for the correct propagation of data will have higher criticality. The criticality values are then normalized across all the nodes. This information can be automatically collected by means of actual traffic analysis – if the network has already been deployed – or through simulations of data communications on the topology and chosen protocol – if the network is still in its design stage. In the second step the network administrators can tune any criticality value by either *(i)* overriding the computed value with a new, custom one, or *(ii)* by specifying "multipliers" for specific areas of the network, to be applied to all the nodes located in that area. The second step is not required. However our two-step approach helps in characterizing the network in the best possible way by leveraging the administrator's knowledge of the intrinsics of the network and its functionality, while still automating most of the activities for characterizing the node criticality.

Our two-steps approach is well suited to many IoT scenarios. First of all, the correct transmission of data is often at the core of the security goal. In these cases the first step of our approach properly addresses the needs to distinguish nodes basing on their in/out-going traffic rate. Second, many IoT networks are designed with a hierarchical structure, thus some nodes need to be considered at a different level of criticality with respect to the other ones. Is the case of cluster-heads in cluster-based network topologies, that are responsible of the correct functioning of nodes belonging to their cluster. Or else, in a home IoT environment it is often the case that external attackers launch attacks trough the router linking the IoT network with the Internet. In such cases it is mandatory to assign the highest criticality value to the router, which represent an access point for intruders.

3.4.2 Overview of the Pareto Analysis

Pareto analysis [93] is a classic optimization method used in situations in which there are multiple competing objectives that must somehow be satisfied simultaneously. The basic idea behind Pareto optimization of three competing objective functions ϕ_1, ϕ_2 and ϕ_3 subject to a set $\mathcal{C}$ of constraints is as follows. Suppose that $\sigma = (x_1, x_2, x_3)$ and $\sigma' = (x'_1, x'_2, x'_3)$ are two different solutions (resource allocation plans) and suppose that ϕ_1, ϕ_2 and ϕ_3 are minimization problems. We say that σ *dominates* σ', denoted $\sigma \rhd \sigma'$, iff:

$$(x_1 \leq x'_1 \wedge x_2 \leq x'_2 \wedge x_3 \leq x'_3) \wedge (x_1 < x'_1 \vee x_2 < x'_2 \vee x_3 < x'_3)$$

A solution σ is said to be *Pareto optimal* w.r.t. a set of minimization problems Φ, and constraints $\mathcal{C}$ if and only if there is no solution $\sigma' \neq \sigma$ such that $\sigma' \rhd \sigma$. We use Pareto analysis to solve the optimization problem defined by Equation 3.1 in order to find a compromise strategy. The main point of this method is the computation of the Pareto curve (see [93]).

3.4.3 First Step: Pareto Analysis for the Defender

To perform our Pareto analysis, let

$$\mathcal{P} = \{(ec(AP), tc(AP), \max_{sr \in AP} crit(sr, AP, N)) | AP \in \mathcal{AP}\}$$

be the set of all possible values for our three objectives.

Definition 3.4.1 (Pareto curve) *The* Pareto curve PC *for the three-objective optimization problem defined by Equation 3.1 is the set* $\{(a, b, c) \mid (a, b, c) \in \mathcal{P}$ *and* $\nexists (a', b', c')$ *such that* $(a', b', c') \rhd (a, b, c)\}$.

ALGORITHM 2:

```
Function(ComputePC (minCR,maxCR)) X = ∅;
for cr = minCR − 1, ..., maxCR do
    tc ← getTC(∞, cr);
    while (tc ≠ null) do
        ec ← getEC(tc, cr);
        X ← X ∪ (ec, tc, cr);
        tc ← getTC(ec, cr);
return Pareto-optimal points in X;
```

where:

$$getEC(tc, cr) = \min_{\substack{AP \in \mathcal{AP} \\ tc(AP) \leq tc \\ \max_{sr \in AP} crit(sr, AP, N) \leq cr}} ec(AP)$$

$$getTC(ec, cr) = \min_{\substack{AP \in \mathcal{AP} \\ ec(AP) < ec \\ \max_{sr \in AP} crit(sr, AP, N) \leq cr}} tc(AP)$$

Algorithm 1 computes the Pareto curve of the three-objective optimization problem defined by Equation 3.1. More specifically, by fixing each possible value of criticality cr (Line 3), the algorithm computes the set X of points $p = (ec, tc, cr)$ (Lines 4-9), where ec and tc are the total energy consumption and the total cost, respectively. The first step is to compute the minimum total cost without considering the energy consumption (Line 4). This optimization is performed by the function $getTC$, and the

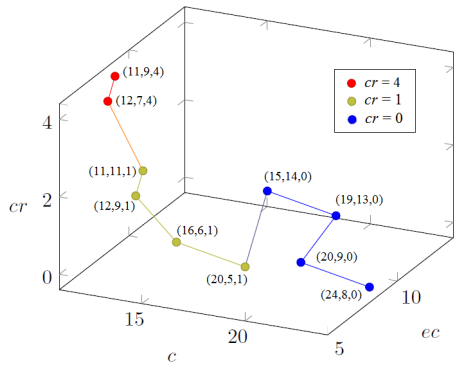


Figure 3.2. An example of Pareto curve.

value is assigned to the variable tc. Then, given tc and cr, function $getEC$ computes the minimum total energy consumption ec (Line 6), and the point (ec, tc, cr) is added to X (Line 7). Now the algorithm computes a different value of tc (Line 8). This value is obtained by the function $getTC$, which minimizes the total cost by imposing that the total energy consumption is strictly less than the one previously computed. If such value exists, the algorithm continues to generate more points by iterating in the while loop, otherwise it exits. Finally, in Line 11 the algorithm returns all Pareto (non-dominated) points of X. Notice that in Line 3, we start to enumerate the values of cr from $minCR - 1$, so that, as no node exists with $cr < minCR$, the plan being calculated with $cr = minCR - 1$ provides $risk = 0$, i.e., each network node is covered by at least two security resources. For allocation plans with $cr = 0$ it is not necessary to perform the second optimization step, since we already know that the risk will be also equal to 0. Figure 3.2 shows an example of Pareto curve, computed for the Pareto analysis of one of the topologies used for our experiments, and discussed in Section 3.6. It can be noticed that no point (ec, c, cr) of the curve is dominated, according to the definition in Section 3.4.2.

3.4.4 Linear Constraints for the Pareto Analysis

We formalize the set of basic constraints as follows.

Variables

- $x_1, \ldots, x_{|\mathcal{R}|}$; if resource sr_i belongs to AP, then $x_i = 1$, otherwise $x_i = 0$;
- $z_{11}, \ldots, z_{\lceil |\mathcal{L}||\mathcal{R}| \rceil}$; if location $loc_i \in Dom(sr_j)$ and $sr_j \in AP$, then $z_{ij} = 1$, otherwise $z_{ij} = 0$;
- $q_1, \ldots, q_{|V|}$; if $loc_i \in L_V$ belongs to the domain of no more then one security resource, then $q_i = 1$, otherwise $q_i = 0$;
- $l_1, \ldots, l_{|E|}$; if link e_i is secure, then $l_i = 1$, otherwise $l_i = 0$.

All variables are binary. The set of basic constraints is shown in Figure 3.3. Constraint 3.4 says that no more than one security resource can be placed on the same location.[2] Constraint 3.5 says that variable z_{ij} is assigned value 1 if $sr_j \in AP$ and $loc_i \in Dom(sr_j)$. Constraints 3.6 and 3.7 capture the fact that, if a location $loc_i \in L_V$ is guarded by no more than one security resource, then $q_i = 1$, otherwise $q_i = 0$. Here, M is a constant big enough to satisfy the constraints for all values of q_i and the lhs of the equation (any integer greater than $\mathcal{R}$). Variables q_i are used to compute the risk and the criticality associated with each resource $sr \in AP$. Constraint 3.8 captures the definition of secure node, according to which a node is secure if conditions (α) or (β) of Definition 3.2.4 hold, i.e., expressions $\sum_{sr_j \in Res(v_i.loc): sr_j \in T_1} x_j$ or c_i, respectively, equal 1. The value of variable c_i is computed by constraints 3.9 and 3.10, and it has value equal to 1 if all links belonging to $Edges(v_i)$ are secure, otherwise it is equal to 0. The definition of secure link is captured by Constraint(s) 3.11, according to which a link is secure ($l_k = 1$) if conditions (σ) or (γ) of Definition 3.2.3 hold, i.e., variables a_k or b_k equal 1. The values of these variables are computed by Constraints

[2]Constraint 3.4 does not bear any loss of generality since a security resource can implement more than one security function.

$$\sum_{sr_i \in Res(loc_j)} x_i \leq 1, \quad \forall loc_j \in \mathcal{L} \tag{3.4}$$

$$z_{ij} = x_j, \quad \forall sr_j \in \mathcal{R}, \forall loc_i \in Dom(sr_j) \tag{3.5}$$

$$\sum_{sr_j : loc_i \in Dom(sr_j)} z_{ij} \leq q_i + M \cdot (1 - q_i), \quad \forall loc_i \in L_V \tag{3.6}$$

$$\sum_{sr_j : loc_i \in Dom(sr_j)} z_{ij} \geq 2 - q_i, \quad \forall loc_i \in L_V \tag{3.7}$$

$$\sum_{sr_j \in Res(v_i.loc) : sr_j.t \in T_\alpha} x_j + c_i \geq 1, \quad \forall v_i \in V \tag{3.8}$$

$$\sum_{e_k \in Edges(v_i)} l_k < c_i + |Neighbors(v_i)|, \quad \forall v_i \in V \tag{3.9}$$

$$\sum_{e_k \in Edges(v_i)} l_k \geq c_i \cdot |Neighbors(v_i)|, \quad \forall v_i \in V \tag{3.10}$$

$$l_k \geq a_k, \quad l_k \geq b_k, \quad l_k \leq a_k + b_k, \quad \forall e_k \in E \tag{3.11}$$

$$\sum_{sr_j : \{v.loc, v'.loc\} \in Dom(sr_j) \wedge sr_j.t \in T_\sigma} d_{kj} \geq a_k, \quad \forall \{v, v'\}_k \in E \tag{3.12}$$

$$a_k \geq d_{kj}, \quad \forall \{v, v'\}_k \in E, \forall sr_j : \{v.loc, v'.loc\} \in Dom(sr_j) \wedge sr_j.t \in T_\sigma \tag{3.13}$$

$$\sum_{loc_i \in \{v.loc, v'.loc\}} z_{ij} \geq 2 d_{kj}, \quad \forall \{v, v'\}_k \in E, \forall sr_j : \{v.loc, v'.loc\} \in Dom(sr_j) \wedge sr_j.t \in T_\sigma \tag{3.14}$$

$$\sum_{loc_i \in \{v.loc, v'.loc\}} z_{ij} < d_{kj} + 2, \quad \forall \{v, v'\}_k \in E, \forall sr_j : \{v.loc, v'.loc\} \in Dom(sr_j) \wedge sr_j.t \in T_\sigma \tag{3.15}$$

$$\sum_{loc_i \in \{v.loc, v'.loc\}} z_{ij} \geq 2 b_k, \quad \forall \{v, v'\}_k \in E, \forall sr_j : \{v.loc, v'.loc\} \in Dom(sr_j) \wedge sr_j.t \in T_\gamma \tag{3.16}$$

$$\sum_{loc_i \in \{v.loc, v'.loc\}} z_{ij} < b_k + 2, \quad \forall \{v, v'\}_k \in E, \forall sr_j : \{v.loc, v'.loc\} \in Dom(sr_j) \wedge sr_j.t \in T_\gamma \tag{3.17}$$

$$z_{ij} + q_i \leq 1, \quad \forall loc_i \in L_V, \forall sr_j \in \mathcal{R} : sr_j.t \in T_\alpha \wedge sr_j.loc = loc_i \tag{3.18}$$

$$\sum_{sr_j : loc_i \in Dom(sr_j)} z_{ij} \geq 1, \quad \forall loc_i \in \mathcal{L} \tag{3.19}$$

Figure 3.3. Basic constraints.

3.12-3.15 and 3.16-3.17, respectively. Constraint 3.18 says that for tamper resistant nodes the variable q_i is forced to 0, i.e., it does not affect the risk and criticality values of the allocation plan is being computed. Finally, Constraint 3.19 says that each location has to belong to the domain of at least one security resource in AP. This last constraint is fundamental because it addresses the dynamic characteristic of IoT networks. In an IoT environment, as new devices can enter and leave the network

$$getEC(tc, cr) = \textbf{minimize} \sum_{sr_i \in \mathcal{R}} sr_i.e \cdot x_i$$

subject to: basic constraints, and

$$\sum_{sr_i \in \mathcal{R}} sr_i.c \cdot x_i \leq tc \tag{3.20}$$

$$v_i.cr \cdot q_i \leq cr \quad \forall v_i \in V \tag{3.21}$$

Figure 3.4. Linear program for computing the minimum energy consumption.

area, it is important to monitor all locations where any new device can move over time. When this happens, we are sure to have at least one security resource monitoring its location. We emphasize that the concept of dynamic topology we refer to is different from the concept of mobility. In our network model it is expected that the nodes are static, and that the topology changes due to the (dis)appearance of nodes. For mobile device scenarios a different approach to resource allocation should be adopted. In Constraints 3.8 and 3.14-3.18 the symbols T_α, T_σ and T_γ are subset of T, the set of security resource types. In particular, T_α is the set of types of security resources that make a node tamper resistant, T_σ is the set of types of security resources that behave as watchdogs, and T_γ is the set of types of security resources that make a node able to establish a security communication channel with its neighbors. In Section 3.5 we implement the proposed method for two real-case IoT scenarios where we show how to compute such sets.

The basic constraints shown in Figure 3.3 capture the dependencies between the variables in the integer linear programs shown in Figures 3.4, 3.5 and 3.6. The objective function of the formulation in Figure 3.4 says that we want to minimize the total energy consumption of the security infrastructure. Constraint 3.20 says that the total cost must be lower or equal to tc, and Constraint 3.21 says that the criticality value of the nodes protected by no more than one security resource, must not exceed cr. The objective function of the linear program in Figure 3.5 minimizes the total

$$getTC(ec, cr) = \textbf{minimize} \sum_{sr_i \in \mathcal{R}} sr_i.c \cdot x_i$$

subject to: basic constraints, and

$$\sum_{sr_i \in \mathcal{R}} sr_i.e \cdot x_i < ec \tag{3.22}$$

$$v_i.cr \cdot q_i \leq cr \quad \forall v_i \in V \tag{3.23}$$

Figure 3.5. Linear program for computing the minimum cost.

cost of the set of security resources. Constraint 3.22 imposes that the total energy consumption be strictly less than ec, while Constraint 3.23 is equal to Constraint 3.21.

3.4.5 Second Step: Best Defender Strategy

Once we have computed the Pareto curve PC, we can choose the point $p^+ = (ec^+, tc^+, cr^+) \in PC$ closest to our requirements of energy consumption, cost, and criticality. p^+ identifies the set AP^+ of different allocation plans that have an energy consumption equal to ec^+, a cost equal to tc^+, and a criticality value equal to cr^+. Among all plans in AP^+ we choose the one which minimizes the risk, i.e., the maximum number of unprotected nodes an attacker can exploit after compromising one security resource. To do that, we solve the optimization problem defined by Equation 3.2, and formalized as shown in Figure 3.6. The intuition behind constraint 3.24 is that instead of bounding the risk associated with each security resource $sr \in AP$ on the right-hand side of this constraint, we set it to an unknown value h and then require the objective function to minimize h. Here, M is a constant big enough to ensure the satisfaction of the equation for all values of q_i and x_j (any integer greater than $|V|$). Constraints 3.25, 3.26 and 3.27 impose that the allocation plan must be

$$getR(ec, tc, cr) = \textbf{minimize } h$$

subject to: basic constraints, and

$$\Big(\sum_{loc_i \in \{Dom(sr_j) \cap L_V\}} q_i\Big) - (1 - x_j) \cdot M \leq h \qquad \forall sr_j \in \mathcal{R} \tag{3.24}$$

$$\sum_{sr_i \in \mathcal{R}} sr_i.e \cdot x_i \leq ec \tag{3.25}$$

$$\sum_{sr_i \in \mathcal{R}} sr_i.c \cdot x_i \leq tc \tag{3.26}$$

$$v_i.cr \cdot q_i \leq cr \qquad \forall v_i \in V \tag{3.27}$$

$$h \geq 0 \tag{3.28}$$

Figure 3.6. Linear program for computing the minimum risk.

computed so to have an energy consumption, a cost and a criticality value not greater than ec, tc and cr, respectively.

3.5 Examples of Real Case Scenarios

In this section we show how to implement the proposed approach for two real case IoT scenarios. The cases we address here are: *(1)* a wireless sensor network exposed to several kinds of attacks; and *(2)* a domestic IoT environment exposed to jamming attacks [89], according to which an adversary conducts radio interference on the links, in order to partially or entirely disrupt a node's signal.

Case 1. To make sure to counteract different kinds of attacks, the defender should secure both nodes and links. Suppose that the available security tools are: an intrusion detection system (IDS), a highly sensitive transceiver (HST)[3], and a tamper resistant sensor node (TRN). The defender can use tamper resistant sensor nodes in place of normal nodes (node hardening), in order to prevent the attacker from having free access to node's memory, and install an IDS on (additional) sensor

[3]A HST increases the device's communication range, so it can interact with more network nodes.

nodes for detecting and stopping ongoing attacks (link/node monitoring). Overall, the defender has available 6 types of security resources as a combination of a set of security tools and a (additional) sensor node:

1. an IDS installed on a network node;
2. an IDS installed on a network node equipped with a HST;
3. a TRN to put in place of a network node;
4. an IDS installed on a TRN to put in place of a network node;
5. an IDS installed on an additional node to deploy in the network area;
6. an IDS installed on an additional node equipped with a HST to deploy in the network area;

The defender assigns to the nodes a criticality value based on the amount of their in/out-going traffic. For this scenario we have that: $T = \{1, 2, \dots 6\}$; for each security resource sr such that $sr.t \in T' = \{1, 2, 3, 4\}$, $sr.loc \in L_V$; for each security resource sr such that $sr.t \in T'' = \{5, 6\}$, $sr.loc \in L_A$; $\mathcal{R} = T' \times L_V \cup T'' \times L_A$; $T_\alpha = \{3, 4\}$, $T_\sigma = \{1, 2, 4, 5, 6\}$ and $T_\gamma = \emptyset$.

Case 2. In the majority of cases a jammer is an attacker that uses its own device for disrupting nodes signal. The defender thus chooses to adopt the link monitoring technique to detect the attack. The IoT devices used in this scenario are not user-programmable; thus, the defender can only adopt additional devices for security purposes (security resources of type 5 and 6 of the list of Case 1). The node's criticality is assigned customly. For this scenario we have that: $T = \{5, 6\}$; for each security resource sr, $sr.loc \in L_A$; $\mathcal{R} = T \times L_A$; $T_\alpha = T_\gamma = \emptyset$, $T_\sigma = T$. Notice that here the defender is only adopting the link monitoring technique, which formally means that conditions (α) and (γ) are false, while conditions (β) and (σ) are true. In the linear programming formulation this is formalized as $T_\alpha = T_\gamma = \emptyset$ which, in Constraints 3.8, forces $\sum_{sr_j \in Res(v_i.loc): sr_j.t \in T_\alpha} x_j$ to be 0, and c_i to be 1 ($(\alpha) = false$

Table 3.1.
Main statistics of the topologies used in the experiments, and packet delivery rate provided by each strategy. $\mathbf{V}$ is the set of nodes, $\mathbf{E}$ is the set of edges, $\mathbf{S}$ is the set of source nodes, and $\mathcal{L}$ is the set of locations.

Topology	$\lvert\mathbf{V}\rvert$, $\lvert\mathbf{E}\rvert$, $\lvert\mathbf{S}\rvert$, $\lvert\mathcal{L}\rvert$	AP_c **(min cost)**	AP_{ec} **(min energy c.)**	AP_{cr} **(min crit.)**	AP^* **(best strategy)**
1 - grid 10×10	100, 342, 10, 400	(76, 21, 10, 6) 68.1%	(56, 45, 10, 20) 0.6%	(96, 24, 1, 6) 87%	(96, 24, 1, 4) 91.1%
2 - grid 10×5	50, 157, 7, 200	(52, 13, 2, 4) 81.1%	(32, 27, 7, 12) 53.5%	(51, 17, 1, 4) 95.6%	(51, 17, 1, 3) 95.7%
3 - grid 5×5	25, 72, 5, 100	(28, 7, 5, 4) 19.1.6%	(18, 13, 5, 7) 13.4%	(32, 8, 1, 3) 92.8%	(32, 8, 1, 2) 99.1%
4 - random	63, 147, 8, 390	(76, 21, 8, 5) 57.4%	(51, 38, 8, 9) 41%	(84, 27, 2, 4) 84.8%	(84, 27, 2, 2) 95.6%
5 - random	62, 135, 7, 395	(84, 21, 6, 5) 47.3%	(47, 43, 6, 10) 79%	(80, 24, 1, 6) 89.4%	(80, 24, 1, 3) 97.3%
6 - random	24, 38, 5, 183	(40, 16, 5, 3) 62.3%	(29, 24, 3, 4) 53.7%	(34, 23, 1, 3) 79.1%	(34, 23, 1, 2) 90.6%
7 - random	14, 28, 4, 93	(20, 5, 4, 2) 47.9%	(14, 16, 2, 1) 54.7%	(18, 15, 1, 1) 96.3%	(18, 15, 1, 1) 96.3%
8 - random	13, 21, 4, 86	(19, 9, 4, 4) 57%	(14, 14, 2, 2) 91%	(24, 10, 1, 3) 93.6%	(24, 10, 1, 2) 96%
9 - random	11, 17, 3, 83	(20, 5, 1, 3) 99.9%	(11, 9, 4, 6) 58.5%	(20, 5, 1, 3) 99.9%	(20, 5, 1, 3) 99.9%

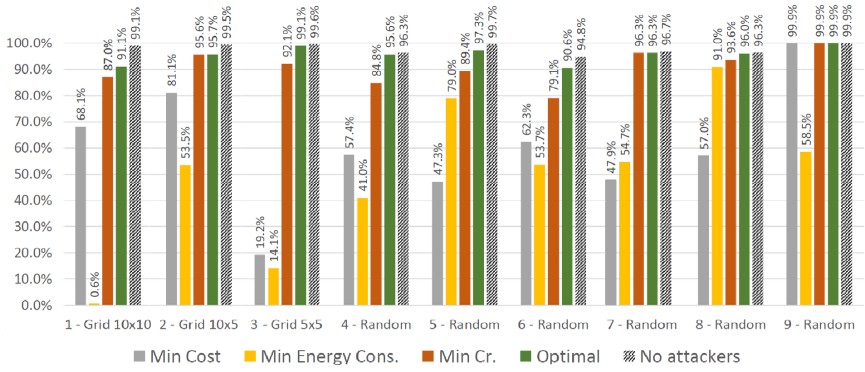


Figure 3.7. Performance of the strategies for each of the cases listed in Table 3.1. Even in an attack-free simulation, the PDR is never 100% due to the packet drop caused by natural physical phenomena and network operations.

and $(\beta) = true$, respectively); and in Constraints 3.11, 3.12, 3.13, 3.16 and 3.17, forces a_k to be 1 and b_k to be 0 ($(\sigma) = true$ and $(\gamma) = false$, respectively).

In both cases, for each security resource, the values of cost c, energy consumption ec and communication range d, depend on the set of adopted tools, and whether an additional node is deployed. For instance, a security resource of type 5, has

$c = c_{IDS} + c_{node}$, $ec = ec_{IDS} + ec_{node}$, $d = d_{node}$; whereas a security resource of type 2 has $c = c_{HST} + c_{IDS}$, $ec = ec_{HST} + ec_{IDS} + ec_{node}$, $d = d_{HST}$.[4]

We assume that the network topology is already known to the security administrator. There are various methods for network topology discovery and we refer the reader to past work [94, 95]. We do not cover such methods in this chapter because of space constraints. Concerning the security resource features, these can be easily determined from their data sheet, where average energy consumption, cost and communication range are usually provided.

3.6 Experimental Results

In this section, we present the experimental results of our proposed framework when applied to the attack scenario described in Case 1 of Section 3.5. The goal is to demonstrate the importance of accounting for effectiveness metrics in the design process of a security infrastructure, besides efficiency. We show how the effectiveness of the best defender strategy is higher compared to any other strategy. We applied our technique to 9 different network topologies, whose main characteristics are reported in Table 3.1. For each topology, we computed four different strategies: *(i)* the one which entails the minimum cost AP_c, *(ii)* the one which entails the minimum energy consumption AP_{ec}, *(iii)* the one which provides a reasonable balance between cost, energy consumption and criticality (c^*, ec^*, cr^*), namely AP_{cr}, and *(iv)* the best defender strategy AP^* (the one which entails the minimum risk, given (c^*, ec^*, cr^*)). We simulated a selective forwarding attack, and measured the packet delivery rate (PDR) of the network for the four strategies.

[4]For resources of type 2, the cost does not depend on the cost of the node, since this already belongs to the network, but the same principle does not apply for the energy consumption, since together a security tool and a sensor node become a single thing.

3.6.1 Settings

We implemented our framework in Java and used IBM ILOG CPLEX 12.5 to solve the ILPs. All computations were run on an Intel Core i7-5600U CPU clocked at 2.59GHz, running Windows 8.1 64bit, with 2GB RAM available for each experiment. To evaluate the effectiveness of the different allocation plans, we implemented the various topologies in TinyOS [47], and carried out 10 independent simulations of 500 seconds each in the TOSSIM simulator [71]. We computed nodes criticality as integer values, ranging from $minCR = 1$ to $maxCR = \lceil \sqrt{|V|} \rceil$. Its computation was based on the amount of data packets DP passing through a node during an attack-free simulation of 500 seconds. More precisely, we normalized the DP value of each node according to a scale from $minCR$ to $maxCR$. Then, we assigned to source nodes the maximum value plus one, in order to assign to these nodes the maximum degree of security. The set of security resources $\mathcal{R}$ is as described in Section 3.5. We assigned a value for cost c, average energy consumption ec and communication range d, as follows:

- network node: $c_n = 2$, $ec_n = 2$, $d_n = 2$.
- IDS: $c_{IDS} = 1$, $ec_{IDS} = 2$;
- TRN: $c_{TRN} = 7$, $ec_{TRN} = 3$;
- HST: $c_{HST} = 5$, $ec_{HST} = 3$, $d_{HST} = 5$.

The values above were assigned so as to reflect real relative differences of cost, energy consumption, and communication range between them. Overall, we believe these values to be realistic for our experimental purposes.

3.6.2 Results Analysis

We simulated a selective forwarding attack [89] (with 50% probability of dropping packets) for 500 seconds, over the nodes no longer protected when the security resource

that maximizes the risk is removed from the defense strategy. Table 3.1 shows the simulation results for each strategy and network topology. We report cost c, energy consumption ec, maximum criticality cr and maximum risk r provided by the strategy, in the form (c, ec, cr, r), and the PDR of the network under attack. Figure 3.7 shows a comparison of the performances for each set of strategies, together with the PDR of an attack-free scenario.

First of all, we note that the most efficient strategies AP_c and AP_{ec} are not the most effective; in fact, due to the higher value of cr and r, they provide a lower PDR with respect to AP_{cr} and AP^*, i.e., the strategies computed by taking into account effectiveness metrics too. Furthermore, we observe that the PDR is inversely proportional to the risk and criticality values, since the more relevant and numerous the unprotected nodes are, the more the lost data packets are. In more detail, in Case 1 AP_{cr} and AP^* provide the same criticality, but AP^* entails a lower risk and, as a consequence, a higher PDR. In Case 2, we observe that AP_c and AP_{cr} provide the same risk, but AP_c entails a lower PDR since its criticality is higher. In Case 7, we have that AP_{cr} and AP^* are exactly the same, because the set of strategies with $cr = 1$ has cardinality 1, i.e., there exist only one strategy and, consequently, it is also the best one. In Case 9, we have that the cheapest strategy AP_c is also the best strategy for $cr = 1$. Furthermore, notice that we always chose an AP_{cr} with criticality greater than 0, which implies a risk greater than 0, and then be able to provide a more meaningful comparison of the strategies. If we had chosen AP_{cr} always with $cr = 0$, the risk would have also been 0, and AP^* would have always been exactly the same of AP_{cr}. Overall, we note that the quality of a strategy usually increases with the introduction of effectiveness metrics in the computation process. Hence, for all the cases (apart case 9), AP_{cr} results in a higher PDR than AP_c and AP_{ec}, and the best strategy AP^* – the one computed by taking also into account the risk besides efficiency and criticality – is even better than AP_{cr}. We conclude that our approach provides a high qualitative defender strategy, for which the minimum cost is used, and that consumes the minimum amount of energy.

3.7 Scalabililty

We now report some preliminary results about our current analysis on large scale network scenarios. For space reasons we do not show all details, but we briefly survey the method.

For large scale networks, the problem could require a large amount of time to be solved. The time Algorithm 2 takes for computing the Pareto curve depends on many factors: the size (number of variables) of the problem, the characteristics of the security resources, the number of solutions (resource allocation plans) found by the algorithm, the adopted ILP solver (for our implementation we used CPLEX [96]), the implementation of the framework and the computational power of the machine used for running it. The size of the problem is computed as: $|\mathcal{R}| + 2|V| + 4|E| + \sum_{sr \in \mathcal{R}} |Dom(sr)|$, where $|\mathcal{R}|$ is the number of x variables, $2|V|$ is the number of q and c variables, $4|E|$ is the number of l, a, b and d variables, and $\sum_{sr \in \mathcal{R}} |Dom(sr)|$ is the number of z variables. We conducted a scalability analysis on a set of five different network scenarios, that differ from each other for topology, set of security resources, and ratio between locations and network nodes. Figure 3.8 shows how the time grows w.r.t. the size of the problem. For all scenarios the set of security resources is that described in Case 1 of Section 3.5, except for $random2$ and $grid3$ where the communication range of the HST is reduced by 20%. The ratio $\mathcal{L}/V$ is 4.5, except for scenarios $grid2$ and $grid3$ where it is 2.25. We based this analysis on two different topologies, grid and random, that are those used in most of the real world cases.

In order to address scalability, we use an approach organized into three steps: *(i)* we divide the network area into smaller sub-areas by using a clustering algorithm; *(ii)* we solve in parallel the sub-problems of security configuration; and *(iii)* we merge the results obtained for each sub-area. We have carried out an initial assessment of this approach and found that, on average, the defender strategies consume +9.11% of energy, costs $+6,83\%$, but reduces the risk by -8.96% of $|V|$, w.r.t. the strategies

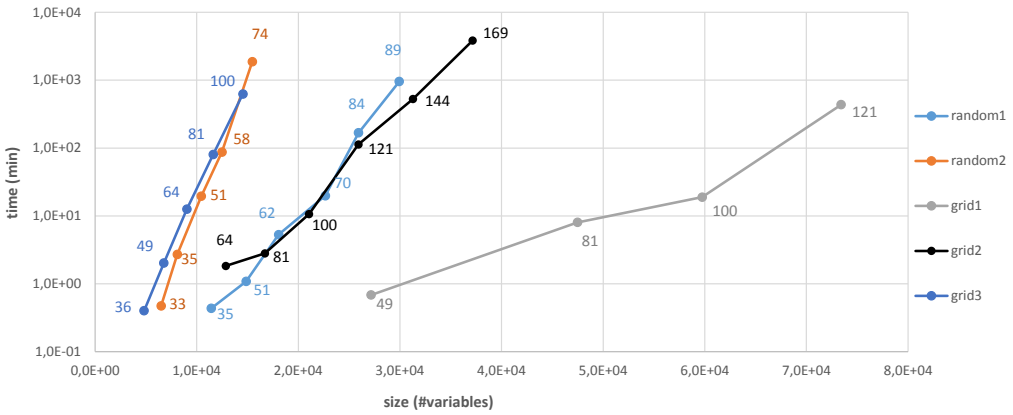


Figure 3.8. How the time grows w.r.t the problem size for 5 different network scenarios. Numbers are the network nodes.

obtained without the preliminary clustering step. For large instances our three steps approach provides solutions with low overhead w.r.t the optimal ones, and due to the use of parallel computing technique, in an amount of time equivalent to that needed for a small instance.

3.8 Security Analysis

Robustness of the Security Allocation. The robustness of the final allocated security configuration depends on the robustness of the security resources chosen and employed by the network administrator. Nevertheless, our approach always provides the best security resource allocation, focusing on the optimal trade-off between redundancy and cost minimization.

Attacker with a Different Strategy. We assume an attacker carrying out the attack that is the worst case attack for the defender, i.e., targeting the security resource that maximizes risk, criticality, or both. In case of an attacker behaving differently than the worst-case scenario (therefore, in some sense, less smartly), the allocation provided by our approach will clearly be able to provide an even higher security than that estimated at the time of the computation of the plan.

Attacks on Several Different Security Resources. No defender strategy is able of protecting the network against an attacker able to compromise all security resources. The strategy that best addresses such a situation is the one that delays the attack as much as possible, by maximizing the number of security tools the attacker needs to compromise for launching the attack. This way, the defender can hope that the attacker will bail out because the "pre-attack" process is much too long or complicated, or that (s)he will be caught while trying to compromise the security tools. The most effective to implement such strategy is to deploy a highly redundant security infrastructure, in order to make any attack very hard to be carried out. However, this solution is not efficient in terms of energy consumption, and can also be quite expensive for large-scale networks. Our approach, instead, provides the most effective solution among the most efficient ones. While this scenario is out of scope for our work, it would be nevertheless easy to add additional redundancy on top of the optimal plan computed, in particular around the areas of the network deemed more important by the administrator.

3.9 Further Uses and Implementations

Our method computes the best defender strategies based on four objectives, that are energy consumption, cost, and criticality, mutually optimized in a first phase (see Section 3.4.3), and risk, minimized in a second phase (see Section 3.4.5). We chose to split the optimization process in those two phases for three main reasons. First, a Pareto analysis with more than three competing goals may require much more time, or alternatively, a much higher computing power in order to be performed. Second, a Pareto analysis involving less then three competing goals provides a much smaller set of solutions, thus limiting the administrator's decisional power. The reason is that the definition of dominated point becomes less restrictive when the number of dimensions decreases (see Section 3.4.2). Finally the choice to optimize criticality in the first phase in place of risk is mainly a matter of convenience, since the range of

criticality values is always much smaller than the range of risk values ($risk \in [0, |V|]$), and thus it is faster to enumerate (see Line 3 of Algorithm 2). However, there are cases in which one or more of the four objectives do not matter for a particular IoT scenario. For instance, in a home environment is often the case that the energy consumption is not a metric of interest since in such small areas the electrical power is always available. Or else, in a one-hop WSN where all nodes perform the same sensing task, there is no criticality. Therefore the second phase is not required as the risk can be evaluated concurrently with the other two objectives, or alternatively, the first phase can be converted in a two-objective optimization process. The cost might be not relevant for small-scale networks and can thus be discarded from the analysis. Furthermore, to restrict the set of solutions of interest, and thus to speed up the optimization process, one can impose bounds on the objectives values. For all this cases it is easy to adapt our framework.

A further use of the proposed method is in the performance assessment of security tools. For a given attack different detection/prevention techniques might exist for which efficiency and effectiveness depend on some feature of the network of interest, such as topology, mobility, routing protocol, etc. A security administrator may want to perform a simulation to determine which IDS/IPS provides the best performance in her(his) specific case. Our framework can also be used in research as an evaluation tool for the comparison of new security techniques with existing ones.

3.10 Related Work

The problem of finding efficient security solutions with the help of game theory and Pareto analysis has been extensively considered in the domain of computer networks [20, 21, 25]. Other work focuses on securing the physical layer from eavesdropping and jamming attacks. In such previous approaches, players include attackers, non malicious users, and the layer itself (with its access control policy). These games are largely based on performance indexes of the physical layer, and the main goal is to

optimize these performance indexes. Approaches in [22, 23] use game-theory to study jamming attacks, while the approach in [24] uses Stackelberg games [81] to model the interaction between defender and eavesdroppers. We like also to mention [97, 98] as typical examples of game theory and Pareto analysis applied to IoT scenarios. The approach in [97] investigates sensor networks in which an attacker can physically capture, replicate the nodes, and deploy sensors into a network, and then proceed to take over the network. A multi-player game is formalized in order to model the non-cooperative strategic behavior between the attackers and the network. The approach in [98] is based on a node clustering algorithm, with effective tax-based sub-carrier allocation tailored to wireless mesh networks with QoS support. Here, Pareto analysis is used for the optimal resource management. Solutions proposed in the context of IoT have focused on efficiency, due to the small "size" of network components, in terms of CPU, memory, and energy budget. Zhou and Chao [99] propose a media-aware security framework for facilitating IoT applications, and provide a design rule and strategy to achieve a good trade-off between system's flexibility and efficiency. Raza et al. [100] propose an IPsec extension of 6LoWPAN, and show that IPsec is a feasible option for securing the IoT in terms of packet size, energy consumption, memory usage, and processing time. Many approaches have been proposed for computing the criticality of graph nodes. Recent approaches target the distributed evaluation and placement of the nodes most critical to network robustness, thus assessing node centrality in a distributed way [101]. Marsden [102] shows empirical evidence that localized centrality measures calculated for one-hop radius neighborhood are highly correlated to the global centrality measure. Kermarrec et al. [103] propose a new centrality measure, called second order centrality, defined in terms of the standard deviation of the time between visits of a perpetual random walk to each node. Arulselvan et al. [104] propose critical nodes to be detected as those whose deletion results in the minimum pairwise connectivity among the remaining nodes. These techniques are useful when the IoT-based system is not deployed yet, but we can still determine

its actual network topology and estimate the nodes criticality with the help of graph theory or, as alternative, simulation tools [71].

To the best of our knowledge, the problem of finding the optimal security resource allocation plan for IoT networks has never been investigated. Past work on IoT security focuses on protection mechanisms against specific attacks [22,23,85], investigates ISO/OSI layer-related security problems [105], or proposes architectures for intrusion detection, attack prevention, or recovery systems [26, 82]. To assure the correct function of IoT networks even under attack, one may need to deploy many such security mechanisms. Therefore, a formal methodology for allocating such mechanisms is needed. To this end, this work proposes a method capable to function for all the different security techniques adopted by the systems cited above.

3.11 Summary

In this chapter, we presented a game-theoretic model to answer the following question: *In an IoT scenario, given a set of security resources and a set of attacks to protect against, which resources should a security manager choose, and how should (s)he allocate them in the network in order to ensure protection with the minimum cost, the minimum energy consumption, and a certain degree of robustness against attacks?* We provide a method for computing the best defender strategy, corresponding to the resource allocation plan that best fits efficiency and effectiveness requirements. With the mutually competing goals of efficiency and effectiveness, we formulate the problem as a Pareto optimization, show how to formulate the defender's problem as a linear optimization problem, and suggest a number of measures to formalize efficiency and effectiveness aspects of the defender's strategy. Our experimental results prove that our method provides the best defender strategy compared to other strategies that do not take into account (all) effectiveness measures, but efficiency only.

The knowledge about the security resources deployed and their placement provided by OptAll can be useful to perform more effective monitoring of the now-deployed

network. In the next chapter, we present our system for detecting attacks in the network that was designed specifically to make the best use of such knowledge.

4 KALIS: A SYSTEM FOR KNOWLEDGE-DRIVEN ADAPTABLE INTRUSION DETECTION FOR THE INTERNET OF THINGS

Following the hardening of the devices and the deployment of network nodes and security resources, according to a strategically-designed allocation, it is crucial to perform continuous monitoring to detect attacks and anomalies. Cryptographic techniques for the IoT [106–110] help in ensuring confidentiality and authenticity of in-network traffic; however such techniques are not able to protect against all attacks. Therefore, it is critical to re-design fundamental security tools for the specific IoT settings. One such tool is represented by Intrusion Detection Systems (IDS). IDSes are vital to maintain the IoT functional. Detecting an undergoing attack is the first line of defense for such *always-on* systems; however, most devices lack logging and reporting mechanisms present instead in enterprise security products [111]. Moreover, an accurate diagnosis is critical for an effective response action.

Different approaches can be taken when designing an IDS for the IoT; however limited research has been carried out on this topic. Most current solutions aim at deploying a custom IDS on each device or group of devices [26, 112]. This approach has the major drawback of being "too local", meaning that each IDS will only have local view of the security situation and insufficient information. Moreover, it does not account for the interoperation of separate IoT devices or groups of devices. Last, it delegates security to the manufacturers of the individual devices. On the other hand, while a more global solution can protect from attacks at a more general level, a preset standalone IDS would not be flexible enough against the complex and heterogeneous IoT ecosystem. Also, simply adapting an existing IDS, designed for traditional computing systems and networks, is not a viable solution. Approaches such as full network scanning to look for threats – used by the widely adopted products such as

SNORT [113] – would not be a good choice for IoT [114] since most emerging IoT standards are shifting to IPv6 [115].

There are, however, several characteristics of IoT that can be leveraged to design an IDS well fit for IoT. First, while heterogeneous, most IoT devices communicate and operate on standard mediums and protocols (such as IEEE 802.15.4 [116], WiFi or Bluetooth for mediums, and ZigBee [117] or 6LoWPAN [118] for protocols). Therefore, as long as a device is able to communicate by using several of these mediums and protocols, effective techniques such as promiscuous overhearing and watchdog-based mechanisms [119, 120] can be deployed. If such a device were able to host a modularly-designed IDS, new detection capabilities could be added as soon as new communication interfaces are available. Moreover, when simply observing events that may be symptoms of security incidents, specific network features can help in ruling out particular attacks, removing ambiguity, improving accuracy, and increasing detection performance. We leverage these observations in the design of our system.

In this work, we first analyze the different attack scenarios that make IoT a unique domain, and investigate the relationship between different network and devices features and related attacks. Then, we propose Kalis[1], a self-adapting, knowledge-driven IDS for IoT able to detect several attacks in real time across IoT systems running different communication protocols and with different security goals. Kalis autonomously collects knowledge about the features of the monitored network and entities, and leverages such knowledge to dynamically configure the most effective set of detection techniques. To the best of our knowledge, Kalis is the first comprehensive approach to intrusion detection for IoT that does not target an individual protocol or application, and adapts the detection strategy to the specific network features.

To minimize the impact on performance and to support IoT devices to which new software cannot be added, Kalis can be deployed as a standalone tool on a separate, external device, providing "security-in-a-box". In that setting, Kalis does not require changes to existing IoT software and has no performance impact on the applications

[1]Kalis is acronym of **K**nowledge-driven **a**daptable **l**ightweight **i**ntrusion detection **s**ystem, but is also a traditional double-edged Filipino sword.

running on the IoT devices. Kalis can be easily extended for new emerging protocol standards and can leverage a customizable library of intrusion detection techniques. Last, it provides a knowledge sharing mechanism that enables collaborative incident detection, and can act as a data source for multi-source security information and event management (SIEM) systems [121].

The contributions of this work are the following:

- the introduction of an IoT attack taxonomy and extensive analysis of the relationship between potential attacks and network/device features;
- the conceptual modeling of a knowledge-driven intrusion detection approach, and its application in the design of a self-configuring, knowledge-driven IDS for the IoT;
- the development and evaluation of a complete IDS prototype, with modules covering a wide range of network features and attacks.

4.1 Background

In this section, we provide an overview of the IoT paradigm and its characteristics, and we introduce some common characteristics of IDSes.

4.1.1 IoT

The IoT paradigm has several characteristics that make it a unique and challenging domain for security measure design. First, the wide range of hardware used for consumer and enterprise IoT devices results in a diverse set of computing capabilities as well as in terms of communication mediums utilized. Second, while traditional computer networks – for which common IDS techniques have been developed – run mostly on top of the TCP/UDP and IP protocols, the IoT depends on a much more diverse set of communication protocols. In some cases, a same device needs to communicate through multiple protocols and interfaces at the same time in order to perform

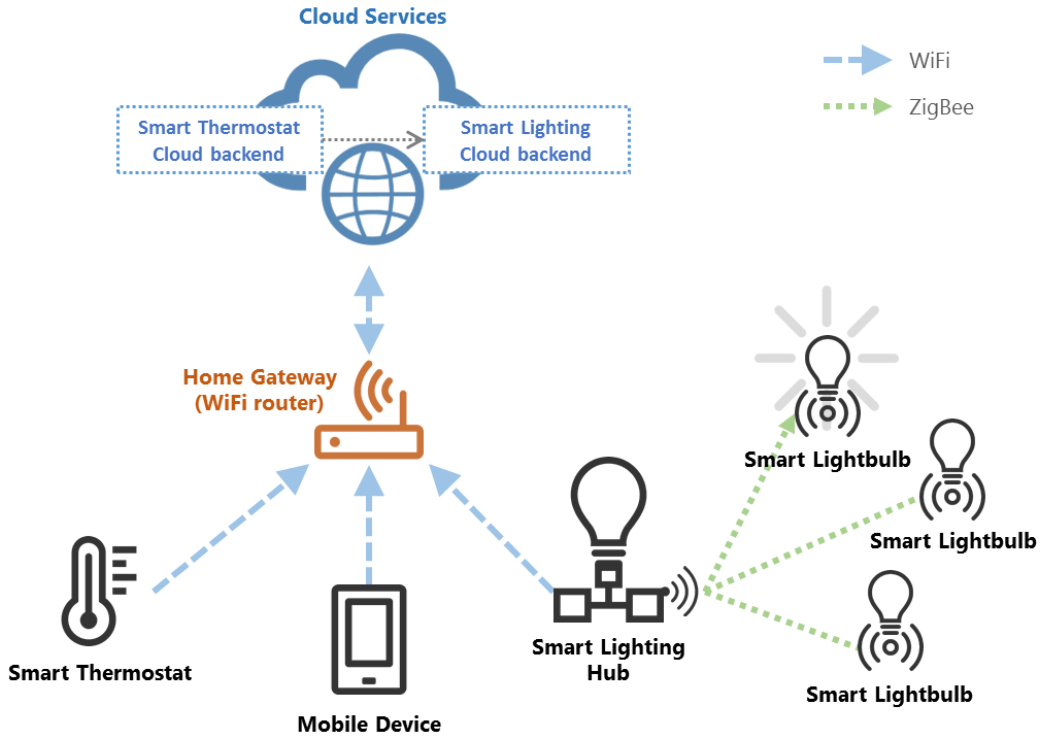


Figure 4.1. A common home automation scenario, depicting the different patterns of IoT communication.

its tasks. In this regard, the communication patterns in IoT are also various. *Device-to-device* communication among different IoT devices (also referred to as "things") – often from different manufacturers – is almost always carried out over the Internet, through cloud services designed specifically for device interoperability. At the same time, though, groups of devices collaborating for a common task – usually from the same manufacturer and part of the same product – form a "master-slaves" structure that we refer to as *hub-to-subs*; in this case, a more powerful device coordinates, controls, and communicates with several more constrained devices through wireless protocols that are more constrained in power or bandwidth, such as IEEE 802.15.4 or Bluetooth.

Consider for example a common home automation scenario (see Figure 4.1) composed of a smart lighting system and a smart thermostat, all controlled via a smart-

phone. The smart lighting system typically consists of a IoT device serving as hub and connected to the Internet through the router, and a set of wireless-enabled light bulbs powered by more constrained microprocessors. When the command of turning on a light is issued from the smartphone, it hits the Cloud services, reaches the hub device through the Internet, and is then propagated locally via a ZigBee-like protocol to the actual light bulb. Conversely, interoperability between separate systems is not usually achieved via local communication. When the smart thermostat becomes aware that the user has arrived home, it can set the correct temperature and also require the smart lighting system to turn on the lights. Even though both devices are in the same household, such communication is typically achieved with the thermostat pushing a command to its own cloud service, then having the cloud services of the two systems communicating, and finally having the smart lighting system's cloud service propagating the command to the hub device.

4.1.2 Intrusion Detection Systems

Most IDSes have a common structure: a data gathering module that collects data possibly containing evidence of an attack, an analysis module that processes such data to detect attacks, and a reporting mechanism to report attacks. The main differences in design choices for IDSes lie in [122]:

- **Data source:** host-based, network-based, hybrid
- **Detection methods:** signature-based, anomaly-based
- **Time of detection:** online, offline
- **Architecture:** centralized, distributed
- **Environment:** wired/wireless/ad-hoc network, ...

Network-based IDSes perform their tasks by externally analyzing network traffic, whereas host-based IDSes require small pieces of software, called *agents*, to run on the monitored devices themselves.

Traditionally, intrusion detection techniques can be broadly classified into either *signature-based* and *anomaly-based*. Signature-based approaches observe the monitored network and attempt to match a pattern of events or data to known attack signatures; such approaches are generally simpler to develop but cannot detect attacks for which the signature is unavailable. Conversely, anomaly-based techniques monitor network traffic and compare it against an established "normal" baseline – which can be dynamically learned by the system or statically set by the administrator – to detect anomalous behaviors. Such approaches are more versatile, as they can detect unknown attacks, but they are harder to implement and more inaccurate, potentially yielding high false positive rates [123].

4.2 Knowledge-driven Intrusion Detection

IDSes typically leverage a library of several detection techniques. Activating all those detecting techniques guarantees a good coverage against potential attacks, but easily leads to inaccuracy and wasted resources. Consider the situation of attacks that share similar symptoms: to a passive external observer – which is the case for network-based IDSes – such attacks will be indistinguishable. Moreover, processing network events and traffic through all the detection techniques requires an unnecessarily high amount of system resources, and can even introduce delays in the attack reaction.

We observe, however, that some features of the monitored network and entities allow one to rule out specific attacks and anomalies; for example, a selective forwarding attack cannot be carried out in a single-hop network. Collecting knowledge about the network's features thus enables the selection of the optimal set of detection techniques. Such knowledge acquisition can be carried out autonomously by the IDS, thus avoiding the need for providing the IDS with predetermined information about the features of the network. Doing so also removes the configuration burden from the user, often not expert in IoT or network security especially in domestic settings, where IoT is witnessing a wide adoption. Moreover, since IoT is a dynamic environ-

ment, a particular configuration of the IDS that is optimal at a certain point in time might not be optimal anymore later on. Supporting continued security enforcement even in face of environment changes is critical.

4.2.1 Conceptual Model

We develop a conceptual model of our knowledge-driven intrusion detection approach. For that purpose, we now define several key concepts:

- **Observation:** a piece of information gathered by observing the available events (e.g., the used communication protocols, a special forwarding field in intercepted packets, a change in signal strength from a node, ...);
- **Feature:** an intrinsic characteristic of the monitored entities and networks (e.g., multihop vs. singlehop network, mobile vs. static network, powerful vs. constrained devices, ...);
- **Symptom:** a particular case of observation that could be associated with a potential security incident (e.g., data losses or inconsistencies, change in traffic frequency, packet duplication or alteration, packet dropping, ...);
- **Detection Technique:** the detection technique for a known security incident, attack or anomaly, carried out purposefully or not, that should trigger a consequent repair mechanism, such as an alert to the user or an automatic response action (e.g., selective forwarding attack, replication attack, Sybil attack, interference, poor link quality, ...).

With these concepts in place, our knowledge-driven intrusion detection approach follows this conceptual process: *using Observation O, the system can discover a Feature F of the monitored entities and network. Given the knowledge about F, the system can effectively determine which one(s) of Detection Technique $\{D_1, D_2, ...D_n\}$ to activate. When only the right detection techniques are active, they will process*

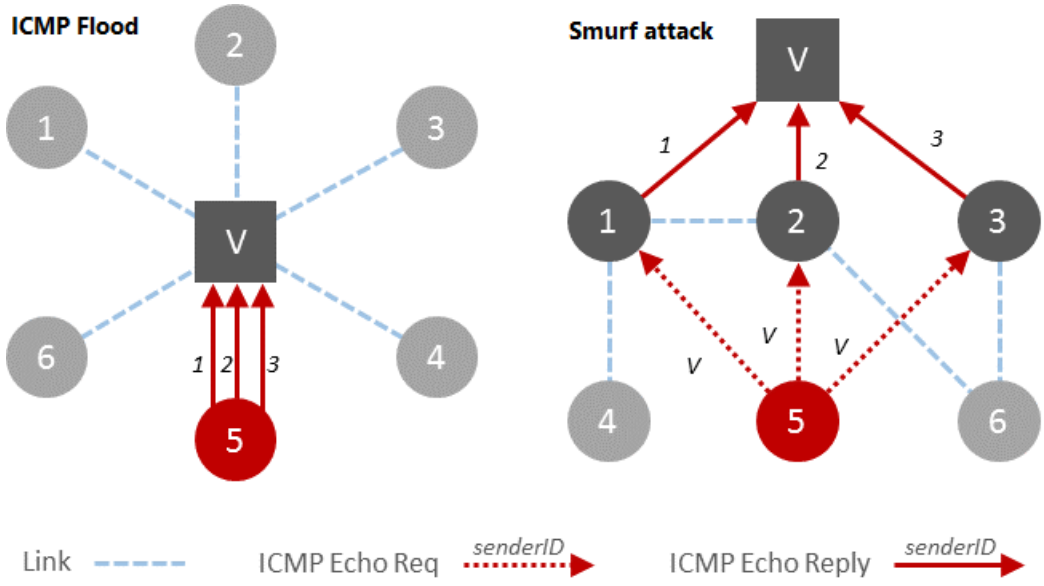


Figure 4.2. ICMP Flood attack vs. Smurf attack.

the available information to identify symptom(s) S, and detect the security incidents happening, improving the accuracy of the system.

Working Example

To illustrate our model, we now walk through a concrete example. We consider two possible attacks: ICMP Flood attack and Smurf attack (see Figure 4.2). In the ICMP Flood attack, a single attacker node sends many ICMP Echo Reply messages to the victim, using several different identities as sender. In the Smurf attack, the attacker sends ICMP Echo Request messages to several neighbors of the victim using the victim's identity as sender; those neighbors will therefore respond with ICMP Echo Reply messages directed to the victim. To an external observer, these two attacks show the same symptoms, that is, a high amount of ICMP Echo Reply messages sent to a victim node. However, the Smurf attack is not possible under single-hop network, and this knowledge can be leveraged to achieve an accurate detection.

Consider the attack and the topology shown in the left-hand side of Figure 4.2. Node 5 will carry out an ICMP Flood attack on victim node V. Our knowledge-driven model can then be instantiated for this example as follows: *observing the traffic, the system can reconstruct the portion of topology in the monitored network, and determine that it is a single-hop network. Given that knowledge, the system activates the detection technique for ICMP Flood attacks and not that for Smurf attacks. Upon the detection of an unusually high amount of ICMP Echo Reply messages to the node, the only active module will unambiguously detect the undergoing ICMP Flood attack.*

4.2.2 Taxonomies

In order for the knowledge-driven model to be effectively employed, it is necessary to formalize and categorize the threats in IoT. We thus propose two taxonomies that look at IoT security threats from different points of view.

Attack Patterns: Taxonomy By Target

In the complex IoT ecosystem, different entities have different capabilities and potential to cause security incidents. We propose a classification and nomenclature from the point of view of attack patterns, considering the source and the destination of each pattern. Table 4.1 summarizes our taxonomy, reporting the attack sources on the rows and the targets on the columns. Figure 4.1 can be used as reference, together with Table 4.1, for a visual interpretation of these attack patterns.

Smart routers[2] are becoming an increasingly available commercial product, and fit well in the IoT ecosystem; therefore, we choose to include them in the categorization not only as target of attacks, but also as potential source of attacks. An interesting aspect of our attack patterns classification is that some source-target pairs are not possible. For example, a sub would not typically be able to attack a router or an

[2]We use the term "router" to indicate both routers and gateways, as they are the same for our purposes.

Table 4.1.
Taxonomy of IoT attacks by target.

		TARGET			
		Internet Service	**Hub**	**Sub**	**Router**
SOURCE	**Internet**	Denial of Service	Remote Denial of Thing	-	-
	Hub	Denial of Service	Control Denial of Thing	Denial of Thing	Denial of Routing
	Sub	-	-	Denial of Thing	-
	Router	-	Control Denial of Thing	-	Denial of Routing

Internet service directly, as it lacks the communication hardware necessary to reach them.

Attacks aimed at Internet services are usually in the traditional form of Denial of Service (DoS) attacks. Recent news have in fact reported the use of IoT devices for botnet attacks [12]. Moreover, we coin the general term *"Denial of Thing"* (DoT) to denote an attack aimed at disrupting the functionality of a *thing*. Some sub-types of DoT attacks are considered in our taxonomy. For example, attacks targeting an IoT hub are typically aimed at denying the execution of some functions of that hub, including the control of all its dependent subs, if any. Therefore, we refer to such attack as a *"Control DoT"*. Last, we use the term *"Denial of Routing"* for all attack patterns that target IoT routers. Such attacks will typically aim at blocking the normal functionality of all the IoT devices on the local network. Note that attacks from the Internet to a local smart router – possible but still far-fetched – cannot be addressed by any local solution, and are therefore out of the scope of our work.

		FEATURES (by class)																																
		deployment		mobility		communication			topology		coverage		composition		QoS		routing protocol			location		availability of prevention techniques											
		one time	iterative	static	mobile	radio	inductive	sound	single-hop	multi-hop	redundant	non redundant	heterogeneous	homogeneous	timeliness	reliability	max power	min energy	shortest path	human av	non human av	crypto puzzle	cryptography	code attestation	tamper-resistant	FHSS	code spreading	identity verification	dynamic routing					
ATTACKS	selective forwarding	•	•	○	○	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	×	•	•	•	×					
	replication	•	•	○	○	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	×	•	•	•	•					
	sinkhole	•	•	○	○	•	×	•	×	•	•	×	•	•	•	•	○	○	○	•	×	•	•	•	×	•	•	•	•					
	sybil	•	•	○	○	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	×	•	•	•	•					
	wormhole	•	•	○	○	•	×	×	×	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	×	•	•	•	•					
	HELLO flood	×	•	○	○	•	×	•	×	•	•	•	•	•	•	•	•	•	•	•	•	×	•	×	•	•	•	•	×					
	ACK spoofing	•	•	•	•	•	×	•	×	•	•	•	•	•	•	•	○	○	×	•	•	•	•	•	•	•	•	•	•					
	data alteration	•	•	•	•	•	×	•	•	•	•	•	•	•	•	×	•	•	•	•	×	•	×	×	×	•	•	•	•					
	data repetition	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	×	×	•	•	•	•					
	transmission delay	•	•	○	○	•	×	•	•	•	•	•	•	•	×	•	•	•	•	•	×	•	•	×	×	•	•	•	•					
	jamming	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	•	×	×	•	•					
	collision	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					
	flooding	•	•	•	•	•	×	•	×	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	•	•	•	•					
	self-propagating code injection	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	×	•	•	×	•	•					
	TCP SYN flood	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					
	smurf attack	•	•	•	•	•	•	•	○	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					
	ICMP flood	•	•	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					
	fraggle attack	•	•	•	•	•	•	•	○	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•					

Figure 4.3. Taxonomy of relationships between IoT network/device features and attacks. Dots and crosses indicate the possibility and impossibility, respectively, of an attack in presence of a specific feature; circles indicate that the appropriate detection technique for the attack depends on the specific feature.

Leveraging Knowledge: Taxonomy By Features

The effective development of a knowledge-driven intrusion detection model requires also a clear understanding of the relationships between monitored network/entity features and security incidents. Therefore, we propose a taxonomy for the most common features and attacks in the IoT. Figure 4.3 shows our classification, with dots and crosses indicating the possibility and impossibility, respectively, for an attack to happen in presence of a specific feature, and circles indicating instances in which the appropriate detection technique for an attack depends on the specific feature.

Note that in many cases the detection technique for a specific attack depends on the characteristics of the network of interest; for instance, for attacks such as sybil, sinkhole, etc., the detection techniques for single-hop networks are significantly different from those adopted for multi-hop networks. Thus, for such attacks, it is important that all and only the appropriate detection techniques are activated depending on the knowledge about the various network features.

Also note that the IoT is unique in being susceptible to a range of attacks that span those proper of constrained WSN nodes as well as those more proper of traditional computer networks. For this reason, our taxonomy includes a very wide set of features to accommodate all potential attacks (even though it is not possible to be exhaustive in the actual instantiation of this classification.). Doing so, it becomes easier to build a comprehensive library of detection techniques that can work well with our knowledge-driven approach. Lastly, note that we include as features also the presence of prevention techniques; for example, cryptographic techniques deployed on some of the monitored devices make the latter immune to attacks such as data alteration.

4.3 Design of Kalis

In this section, we introduce the requirements that have driven the design of Kalis, and then provide a detailed presentation of the event-driven architecture and the components of Kalis.

4.3.1 Design Requirements

The design of an IDS for a domain as complex as the IoT must fulfill several important design requirements. Following are the ones we considered in the design of Kalis.

- **Asynchronous and Event-driven.** Since the IDS has to capture and analyze big amounts of traffic for several possible threats across many data sources and protocols, the IDS should be completely asynchronous and event-driven.
- **No software changes.** As IoT software is often proprietary, closed-source, and heterogeneous, it is not realistic for an IDS to require changes to the source code of application software. Rather, the IDS should be able to monitor network and devices as an external entity through overhearing and environment sensing.

- **Multi-medium and Multi-protocol.** Different IoT devices, applications and products leverage different communication mediums and protocols. Moreover, all of the attack patterns discussed in our taxonomy by target need to be considered. Also, an IDS for the IoT should be able to comply with several such standards, as well as be extensible so to be able to support new technologies or protocol standards that emerge.

- **No performance overhead.** IoT applications have a wide range of purposes characterized by different requirements in terms of Quality of Service (QoS). Therefore, an IDS should have no impact on the performance of the devices' applications.

- **Collaborative.** Interoperation is one of the core characteristics of the IoT paradigm. Moreover, a single point of view is not always sufficient for acquiring knowledge about the environment and detecting security incidents. Thus, an IDS should enable knowledge sharing, as well as collaborative detection techniques.

4.3.2 Architecture

In this section, we discuss the architecture and all the individual components of Kalis (see Figure 4.4).

With respect to the different design choices of IDSes discussed in Section 4.1.2, Kalis is a *network-based*, hybrid *signature/anomaly-based*, *hybrid centralized/distributed*, *online* IDS that adapts to *different environments*.

Communication System

The Communication System acts as interface with the external world. Specialized subcomponents take care of interacting with traffic on different protocols. In our current design we include ZigBee/XBee/6LoWPAN (on IEEE 802.15.4), WiFi (on

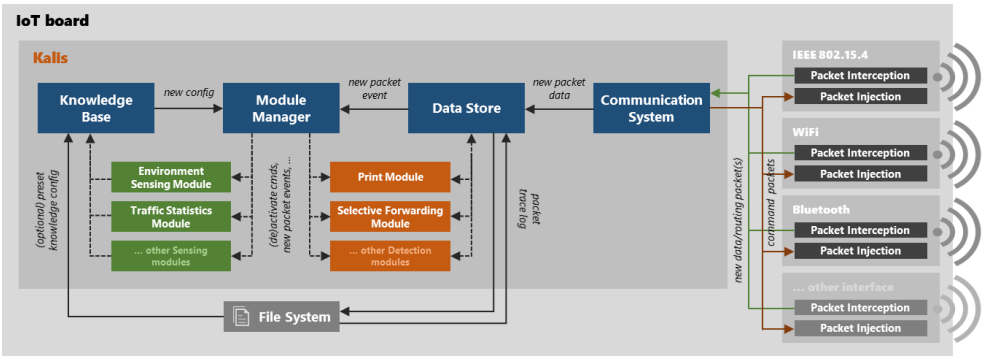


Figure 4.4. The high-level architecture of Kalis.

IEEE 802.11), and Bluetooth traffic, but the design makes it easy to extend to future standards. The Communication System overhears all traffic on all the supported interfaces, satisfying the breadth requirement of considering all attack patterns discussed in our taxonomy by target.

Data Store

All traffic from all sources is managed and stored in the Data Store component. The Data Store listens for events from the Communication System on newly captured packets, manages a history of recent traffic for modules to access, and logs all traffic on disk, if required by the user. In order to appropriately utilize memory, only a sliding window of configurable size of the most recent packets is kept in memory. Logs from disk can also be replayed for traffic analysis by the network administrator in case security incidents are detected. The Data Store abstracts the traffic sources by replaying traffic transparently to the detection modules, which will perform their tasks as if operating on live traffic.

Knowledge Base and Collective Knowledge Management

A key component to Kalis is the Knowledge Base. It stores all the available information about the features of the monitored entities and networks in a unique,

centralized place, as well as it provides this information to all the parties that require it, such as various Detection Modules and the Module Manager. In Kalis, we refer to an individual piece of knowledge as *knowgget* (i.e. "knowledge nugget").

Knowledge Modeling. The pieces of knowledge representing features managed by the Knowledge Base in Kalis can vary significantly with respect to the type of data that they are represented by. For example, the knowledge about a portion of the network being multi-hop can be modeled as a boolean data type, but the knowledge about the total number of nodes monitored by the IDS has to be represented by an integer. For this reason, we choose to keep the model as agnostic and generic as possible, and we model each knowgget as a label, describing the information represented, and its associated value of any type. Additionally, each knowgget has a "creator" field – representing the Kalis node that created it (useful for knowledge sharing, as discussed later in this section) – and an optional "entity" field – in case it is specific to an individual monitored entity (e.g., the detected signal strength for a monitored sensor node). Moreover, some knowggets may also in turn be composed by several different pieces of data; consider for example the knowledge about the current traffic frequency (as packets per second), having several sub-pieces of information for each different packet type, such as TCP SYN, TCP ACK or TinyOS CTP. We refer to these as *multilevel knowggets*. In this case, the label of a multilevel knowggets is not associated with a single value, but with a group of other knowggets, in a tree-like structure. Last, Kalis does not know in advance all the knowggets that will be collected, and new modules may want to store new, previously unknown knowggets. Therefore, the set of labels is not fixed, and is dynamically managed as a multi-level map data structure. Figure 4.5 shows an example of Knowledge Base containing some knowggets about the monitored network.

Formally, a knowgget k is defined as the tuple $k = \langle l, v, c, e \rangle$, where l is the label, v is either a primitive value or a set of knowggets (for multilevel knowggets), c is the identifier for the Kalis node creator of k, and e is the entity related to k (or `null` if none).

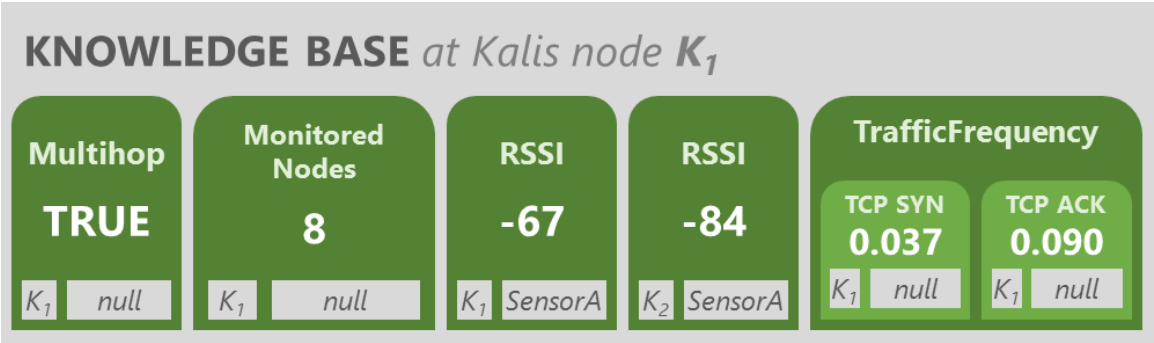


Figure 4.5. An example of knowledge base with heterogeneous knowggets, each showing label, value, creator field, and entity field.

```
KnowledgeBase {
  "K1$Multihop" = "true",
  "K1$MonitoredNodes" = "8",
  "K1$SignalStrength@SensorA" = "-67",
  "K2$SignalStrength@SensorA" = "-84",
  "K1$TrafficFrequency.TCPSYN" = "0.037",
  "K1$TrafficFrequency.TCPACK" = "0.090"
}
```

Figure 4.6. Key-value pair representation of the Knowledge Base in the implementation of Kalis.

Collective Knowledge. The collected knowledge represents the view of the network portions surrounding the Kalis node and, therefore, the latter's specific point of view. In most cases, this knowledge is exactly what is needed to efficiently and effectively perform intrusion detection in that portion of the network. For example, while other parts of the network might have a multi-hop topology, this is not relevant and potentially harmful to an accurate detection as compared to the knowledge that the local area – for which this Kalis node is responsible – is single-hop. In some cases, however, sharing knowledge among different Kalis nodes can enable the detection and discovery of global features useful for intrusion detection. As an example, being aware that other Kalis nodes are noticing changes in signal strength for specific monitored

entities can enable the local Kalis node to correlate such changes with those experienced locally and detect mobility in the network. For this reason, we provide as part of Kalis a mechanism for collective knowledge management. Such mechanism allows sharing and synchronization of selected pieces of information across all Kalis nodes.

To enable this mechanism, a module inserting a knowgget of collective interest into the Knowledge Base can simply mark the knowgget as "collective". The Knowledge Base will then take care of automatically communicating significant changes in that knowgget to the other Kalis nodes for storage in their Knowledge Bases, making sure to mark the appropriate identity in the "creator" field, which reports which Kalis node originally generated and shared the knowgget. Note that this mechanism does not provide a way for a Kalis node to overwrite or alter knowledge in another Kalis node. When a Kalis node, say T_1, receives a new or updated collective knowgget k from a different Kalis node, say T_2, the Knowledge Base of T_1 will check to see if the label *and* creator of k matches any existing knowgget in the Knowledge Base. Therefore, T_1 can only update those knowggets in T_2 that were originally generated by itself.

Static Knowledge. Kalis also provides the user with the possibility of specifying initial or static knowggets about the IoT network. For example, if the network does not have mobility and will always remain so, it makes sense for the user to simply provide Kalis with this information, as it can be leveraged for example to avoid trying to detect mobility in the network, and to improve the detection accuracy as well. For this purpose, the Knowledge Base optionally loads a configuration file from the file system, providing initial settings and a-priori available knowggets. Figure 4.7 shows the (JSON-inspired) grammar for the language used in the specification of configuration files, and Figure 4.8 shows an example of configuration file to activate by default two modules (passing parameters to one of them), and to insert in the Knowledge Base the a-priori knowledge that the network is static. Note that in this case the knowggets might specify an "entity" field, but not a "creator" field, as they will be assigned the local Kalis node's identifier as such.

⟨*config*⟩ ::= ⟨*modules*⟩ ⟨*knowggets*⟩

⟨*modules*⟩ ::= '`modules = {`' ⟨*module-list*⟩ '`}`'

⟨*module-list*⟩ ::= ⟨*module-def*⟩ '`,`' ⟨*module-list*⟩
| ⟨*module-def*⟩

⟨*module-def*⟩ ::= ⟨*module-name*⟩ ['`(`' ⟨*param-list*⟩ '`)`']

⟨*param-list*⟩ ::= ⟨*key-value-pair*⟩ '`,`' ⟨*param-list*⟩
| ⟨*key-value-pair*⟩

⟨*knowggets*⟩ ::= '`knowggets = {`' ⟨*knowget-list*⟩ '`}`'

⟨*knowget-list*⟩ ::= ⟨*key-value-pair*⟩ '`,`' ⟨*knowget-list*⟩
| ⟨*key-value-pair*⟩

⟨*key-value-pair*⟩ ::= ⟨*key*⟩ '`=`' ⟨*value*⟩

Figure 4.7. Grammar for Kalis configuration files.

```
modules = {
  TopologyDetectionModule,
  TrafficStatsModule (
    activationThresh=1,
    detectionThresh=2
  )
}
knowggets = {
  mobility = false
}
```

Figure 4.8. Example of Kalis configuration files.

Modules

Kalis is designed to be fully flexible and extensible. For this reason, any network feature-specific or attack-specific functionality is implemented as independent module. The Module Manager component takes care of coordinating all the modules, activating/deactivating them when necessary, depending on changes in the Knowledge Base, routing new packet events to all the interested parties, and collecting alerts about detected incidents. Each module is capable, given a particular instance

of the Knowledge Base, to determine whether its services are required and, therefore, whether it should be active at that particular point in time.

Kalis includes two different types of modules: Sensing modules and Detection modules.

Sensing Modules. Sensing modules are the core of the autonomous knowledge-discovery mechanisms of Kalis. We include in our prototype a few basic sensing modules, including a Topology Discovery module, a Traffic Statistics Collection module, and a Mobility Awareness module. The first uses the captured traffic to reconstruct the local topology and differentiate between multi-hop and single-hop networks, the second continuously collects statistics about the traffic load, differentiated for each type of packet (e.g., ZigBee data, ZigBee routing, TCP SYN, TCP ACK, ...), and the third leverages the signal strength to dynamically detect whether the network is static or mobile. We provide more details on the implementation of these modules in Section 4.4. Whenever a sensing module finds a relevant change in network features – such as the discovery that a portion of the monitored network is multi-hop – it will store this new knowgget into the Knowledge Base. The Knowledge Base will in turn notify the Module Manager that recent changes to the available knowledge might require revisiting which modules are activated or deactivated.

Detection Modules. Detection modules are actually responsible for analyzing the captured traffic – together with the available knowggets – and detect anomalies and security incidents. Each module is specialized for a specific attack, but some techniques might be able to be generalized to detect attacks with similar symptoms but different severity or root causes – e.g. selective forwarding attack vs. blackhole attack. As discussed in Section 4.1.2, intrusion detection techniques in general are either *signature-based* or *anomaly-based*, and IDSes often operate in only one of the two fashions; however, the large amount of data and knowledge made available by Kalis makes it possible to have a library including detection modules of both kinds, increasing the accuracy in detecting well-known attacks while at the same time being able to react to unknown security incidents as well. In our prototype, we include

several basic detection modules for common attacks, as selective forwarding, SYN flow, ICMP flood, replication, and more.

4.4 Implementation

Development Environment. We implement Kalis using Java on an Odroid xu3 development board. In order to interact with the IEEE 802.15.4 traffic for our prototype, we leverage a TelosB [57] wireless sensor mote with a custom TinyOS [47] application as bridge. Moreover, we integrate into Kalis the `tcpdump` utility – which internally uses the *libpcap* library – in order to promiscuously monitor all WiFi traffic. Our implementation makes use of Java Reflection in various parts of the system. For example, when the configuration file is parsed and a module is specified to be activated, the corresponding class is dynamically instantiated by name. If the configuration file specifies some parameters for that module, Kalis looks for properties in the instantiated module object whose names match the specified parameters, and sets them to the provided values. This implementation makes the entire core of Kalis agnostic of the specific classes that implement the modules, thus making it possible to add new modules without the need to recompile the entire system as long as those modules implement the required interfaces.

Event-driven Architecture. To fulfill the design requirement of being event-driven and asynchronous, all the components in Kalis (see Figure 4.4) run independently. For example, when a new packet is captured on any protocol, all the interested parties are asynchronously notified of the new packet event, and can independently and concurrently process the new information. In the same way, when any of the detection modules detects a potential security incident, it raises a detection event that is then routed to all the subscribed parties. This also allows Kalis to interoperate with cloud-based monitoring dashboards, automated response systems, and real-time user notification mechanisms.

Knowledge Representation. To implement our Knowledge Base, we choose to model each knowgget as a key-value pair, in which both the key and the value are represented as strings. When querying the knowledge base, the modules can either retrieve the raw value and parse it independently, or specify what is the data type they expect in return for a given key and the knowledge base will attempt to parse the string as that data type. We choose an encoding of the key that allows for fast queries. Given a knowgget $k = \langle label, value, creator, entity \rangle$, Kalis encodes it as a key "*creator*`$`*label*`@`*entity*" and a value "*value*". Looking up local (or collective) knowggets only requires searching for the prefix matching (or not matching) the identifier of the local Kalis node. Instead, looking up knowggets related to a specific entity only requires searching for keys with a suffix matching the identifier of the entity of interest. Last, finding a single specific knowgget is done by matching the key exactly. This model also allows Kalis to uniformly represent multilevel knowggets by flattening the hierarchy of labels in dot notation; that is, the sub-information of the "TrafficFrequency" knowgget about TCP SYN packets created by Kalis node T_1 is represented as an individual knowgget with key `"T1$TrafficFrequency.TCPSYN"`. Figure 4.6 shows an example of how the Knowledge Base in Figure 4.5 is represented in our implementation of Kalis.

Collective Knowledge Synchronization. The implementation of the synchronization mechanisms for collective knowledge relies on a few building blocks. First, the discovery of peer Kalis nodes is carried out by means of periodical beaconing on the local network. Each Kalis node will listen for advertisement broadcast packets from other Kalis nodes, and add newly-discovered nodes to a peer list. This is a commonly used *discovery-through-advertisement* pattern that is effective especially for local networks with a moderate number of peers (reasonable assumption for an ideal deployment of Kalis). All communications among the nodes are encrypted, and only enable a one-way communication (in each direction) between pairs of nodes, without the need for interaction beyond the acceptance of incoming new or updated collective knowggets. Our current prototype uses WiFi as communication medium,

but the technique could be adapted to any other medium supported by the hardware. For example, an extension to our prototype could use mainly WiFi but with fallback on Bluetooth in case of connectivity issues through the local WiFi router.

Sensing Modules. As part of our prototype, we include several sensing modules. The Topology Discovery module detects multi-hop and single-hop topology by analyzing the captured traffic. The features used for this analysis include the communication medium used (IEEE 802.15.4 or WiFi), the detection of known protocols (such as RPL in 6LoWPAN or Collection Tree Protocol in TinyOS), the inclusion of specific forwarding/next-hop headers in packets, and more. The range of characteristics that are leveraged to understand the topology of the network can be extended when new protocols or mediums are standardized for the IoT. The Traffic Statistics Collection module maintains detailed statistics about the frequency of the various types of traffic overheard in the network, both on a global and per-monitored-device level. In our implementation, we consider several different types of traffic, including TCP SYN, TCP ACK, ICMP Requests, ICMP Responses, ZigBee plain packets, Collection Tree Protocol packets, and more. For each traffic type, the module keeps track of the number of packets per unit of time (configurable but set to 5 seconds by default); the frequency of each type of traffic is recorded both globally for the whole network, and for each individual monitored device (to support an accurate detection of targeted DoS-like attacks and subsequent potential response actions.) The Mobility Awareness module uses a simple approach that detects mobility when any node's signal strength changes more than a certain threshold. More complex techniques could also be employed, but are out-of-scope for this work.

Dynamic Detection Module Configuration. We implement the dynamic activation and deactivation of detection modules based on the changes in the Knowledge Base via a *publish-subscribe* mechanism, for best efficiency. Each detection module can subscribe to changes on one or more knowggets by key, and is automatically notified; the modules will therefore notify the Module Manager when their services are no longer required, according to the latest knowledge.

4.5 Evaluation

In our experiments, we evaluate (a) the breadth of the IDS coverage over heterogeneous networks, devices and protocols, (b) the benefits of the knowledge-driven approach on the detection accuracy in terms of false positives and detection rate, as well as resource consumption, (c) the reactivity of the IDS in the dynamic discovery of a changing environment, and (d) the benefits of the collaborative knowledge mechanisms.

4.5.1 Experimental Setup

We evaluate Kalis by placing the IDS node near a network of heterogeneous, real-world IoT devices. Our setup includes a small WSN of 6 TelosB nodes, a Nest Thermostat, an August SmartLock, a Lifx smart lightbulb, an Arlo security system, and an Amazon Dash Button. All the WSN nodes are programmed with a TinyOS application that sends a data message every 3 seconds towards a node acting as base station, using the Collection Tree Protocol (CTP) [124]. Concerning the WSN traffic, the Kalis node is placed near the middle portion of the WSN, able to overhear intermediate hops of data packets. Since compromising commodity IoT devices – especially to carry out various controlled attacks in a repeatable way – is very difficult, we choose to record and replay actual traces of network traffic from these devices, enhanced with additional packets representing symptoms of such attacks. For each attack scenario, we run the systems on 50 symptom instances, representing the ground truth for detection. We believe that this setup truthfully represents the complexity of the IoT ecosystem, including both multi-hop and single-hop networks, different protocols on different mediums (CTP on IEEE 802.15.4, TCP/IP on WiFi), and very different devices in terms of computational power and functionality. Furthermore, to simulate potential response actions upon an IDS detection, we program as simple countermeasure the temporary revocation from the network of any node identified as suspect by the IDS.

4.5.2 Benefits of the Knowledge-Driven Approach

In our experiments, we compare our knowledge-driven approach to that of a traditional IDS. For total fairness with respect to the detection techniques, we emulate a traditional IDS by running our system without Knowledge Base, and with all the modules active at all times. We compare the systems on several metrics: (i) **Detection Rate** – number of adverse events detected out of all the adverse events in the test scenario; (ii) **Classification Accuracy** – number of correctly classified attacks out of all the detected attacks; (iii) **Countermeasure effectiveness** – how positive a response action based on the detections of Kalis is for the overall network; (iv) **CPU usage**; (v) **RAM usage**.

ICMP Flood attack on a single-hop network

The first scenario we use for the evaluation is that of an ICMP Flood attack on a single-hop network (see Section 4.2.1). In this setting, the traditional IDS identifies all the attacks (high detection rate); however it generates false positives as it is not able to disambiguate the ICMP Flood attack from a Smurf attack.

With respect to countermeasures, the ICMP Flood detection module considers as suspect all nodes within one hop from the victim, and attempts an approximate disambiguation through a comparison of the signal strength with previous overheard communications [125, 126]; conversely, the Smurf attack detection module considers as suspect all nodes at a 2-hop distance from the victim, and also approximately disambiguates the attacker through signal strength. We observe that, in this scenario, Kalis revokes the correct attacking node, while the traditional IDS attempts to revoke the only node two hops away from the victim, which in a simplistic graph exploration is the victim node itself, therefore disconnecting the entire network.

Replication attack on static vs. mobile network

The replication attack is an application-independent attack unique to wireless networks of constrained devices. In such attack, malicious devices are added to the network as replicas of some legitimate node(s), allowing the adversary to steal sensitive information, disrupt the routing, or inject false data in the network. Many detection techniques exist for this attack; however each one is specific to a network with certain characteristics, e.g. mobility [127]. In this experimental evaluation, we provide two different detection modules for replication attacks, one suitable for static networks, and the other for mobile networks. The network in this evaluation randomly changes between a static and mobile behavior for the nodes over time. We repeat the evaluation 100 times, each time carrying out 3 replication attacks (i.e., sending data packets from 3 nodes that are replicas of legitimate nodes in the network). The traditional IDS randomly selects one of the two modules for each of our experiment runs, simulating closely a static module library configuration by the user that does not adapt to the changes in network features. Kalis, instead, leverages the knowledge provided by the Mobility Awareness module, and dynamically selects modules for the current network mobility setting. We observe the detection rate and accuracy of both Kalis and the traditional IDS approach and, as expected, while Kalis always uses the right modules, the traditional IDS approach misses some attacks when the active module is not the one suitable for the current mobility profile of the network.

Overall Results

Over the two different scenarios presented in Sections 4.5.2 and 4.5.2, we summarize the results in Figure 4.9 for effectiveness metrics, and in Table 4.2 for performance metrics. Kalis achieves 100% classification accuracy, since it always leverages the optimal set of modules for detection based on the knowledge about the network's features. Due to the fact that the detection techniques used cannot always detect all attacks, the detection rate is not perfect; however, this is independent from Kalis, and the

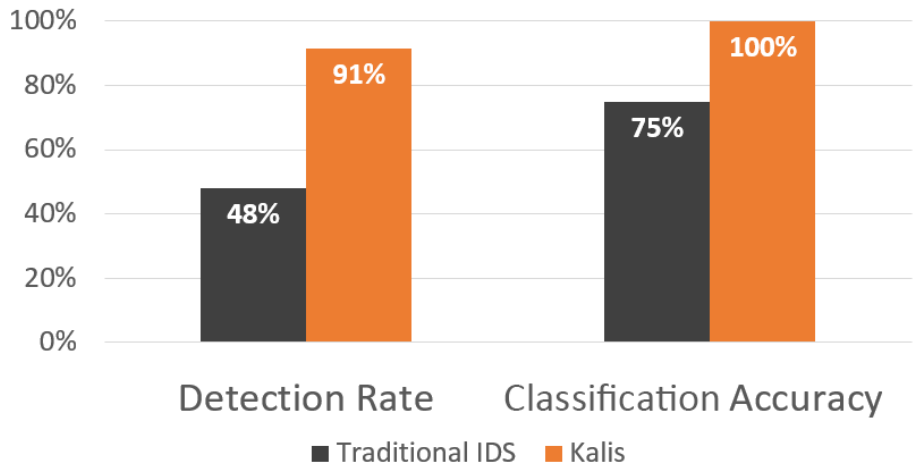


Figure 4.9. Effectiveness comparison for Kalis vs. a traditional IDS approach across all experimental scenarios (averages).

Table 4.2.
Performance comparison for Kalis vs. a traditional IDS approach across all experimental scenarios (averages).

	Trad. IDS	**Kalis**	
CPU usage (%)	0.22%	0.19%	(−16.00%)
RAM usage (kb)	23961.06	13978.62	(−41.66%)

comparison with the traditional IDS approach that uses the exact same detection techniques still shows the benefits of Kalis. The results show that our prototype is very lightweight in terms of CPU and RAM requirements, and that the knowledge-driven approach of Kalis outperforms the traditional IDS on all the considered metrics.

4.5.3 Reactivity to Environment Changes

In order to evaluate the reactivity of Kalis, we start it with a configuration file that does not activate any detection modules by default and does not contain any a-priori knowgget. We then let Kalis monitor a ZigBee network in which one node is programmed to carry out a series of selective forwarding attacks, and measure how soon Kalis detects the first attack. The selective forwarding detection module only

activates upon discovering a multi-hop network; the Topology Discovery sensing module is able to detect such feature from the first CTP packets intercepted. Therefore, we verify that Kalis correctly detects 100% of the selecting forwarding attacks from the very beginning of the communications, even with no detection modules initially active.

4.5.4 Knowledge Sharing

We evaluate Kalis in a scenario in which collaborative knowledge sharing enables the selection of the appropriate set of detection modules, improving the accuracy of the system. In this scenario, two Kalis nodes are monitoring two different portions of a ZigBee network. One node in each portion is malicious, namely nodes B_1 and B_2, and they both collude in carrying out a wormhole attack. In such attack, B_1 does not correctly forward traffic, transmitting it instead directly to B_2. The Kalis node observing the behavior of B_1 would, by itself, detect a blackhole attack, while the Kalis node observing B_2 would, without further information, consider it a source of traffic. However, correlating the events between the two Kalis nodes, they are able to correctly identify such attack as a wormhole.

4.6 Related Work

Extensive research has been carried out in the area of IDSes for traditional networks. Two popular open source IDSes are SNORT [113] and Bro [128]. Both IDes rely on network information gathered using a packet sniffer, and detect attacks using signature matching over this information. Their signature-based schemes provide good detection rates for specified, well-known attacks. However, the techniques used in these traditional systems are not applicable to the IoT domain. For example, techniques such as host scanning or port scanning would be ineffective on most emerging IoT standards that use IPv6 as addressing scheme. Also, both SNORT and Bro only work on traditional networks (wired and wireless), while Kalis supports a wide variety

of mediums and related protocols, and can be easily extended with the emergence of new IoT standards. Several IDSes have been developed for WSN [129–133]; however, they suffer from one or more limitations with respect to IoT: inability to adapt, applicability only to a single platform and protocol, small and specific range of detection techniques, complete dependency on collaboration, reliance on the existence of a central control point.

While much research has been carried out on intrusion detection for traditional systems and WSNs, the work on IDSes specifically tailored to the IoT is still in an early stage. One of the most relevant tool for intrusion detection in IoT is SVELTE [26]. SVELTE is both centralized (at the hub of an IoT system/group of devices) and distributed (at each sub). It is composed of a module called *6Mapper*, which reconstructs the topology of the subs with respect to the hub, and an intrusion detection module, which analyzes data and detects incidents. The IDS is complemented by a mini firewall that filters undesirable incoming traffic from the Internet. In comparison to Kalis, SVELTE (a) is host-based and therefore requires modifications to the IoT devices' software to deploy the IDS component, (b) targets a single IoT system (hub and subs), (c) is primarily designed for devices communicating via the RPL protocol (IPv6 Routing Protocol for Low-Power and Lossy Networks) [134] and cannot be expanded to multiple heterogeneous protocols and mediums, (d) leverages an extensible but predefined set of detection techniques, with no dynamic activation, (e) does not adjust to environmental changes.

Liu *et al.* propose the application of Artificial Immune Systems to IoT IDSes [112,135], mapping concepts between the two domains and presenting a detection mechanism based on datagram signature analysis, with the possibility of sharing *vaccines* among IDS nodes when a new attack signature is generated. While their approach leverages genetic-like algorithms to generate detectors for unknown attacks, it is unclear how to determine the ground truth about legitimate datagrams. Moreover, the administrator is still required to understand which attack a new detector is capturing and supplement the attack library knowledge. While the self-adaptation

mechanisms of the authors' systems aims at providing flexibility as in Kalis, their attack detection mechanism is limited to string-matching. Last, different areas of the network guarded by different Kalis nodes will all have access to the same library of Detection Modules as with vaccines sharing, but Kalis will activate only the necessary ones for the monitored network portion, improving efficiency and accuracy. Jun *et al.* propose the use of Complex Event-Processing (CEP) techniques for intrusion detection in IoT [136]. In CEP, a stream of events is filtered through queries to select those relevant for attack detection. Such work focuses on improving the IDS performance online rather than offline. Our work leverages similar event-driven techniques, but uses the Knowledge Base to avoid the processing of unnecessary rules.

Some research efforts have focused on developing a taxonomy of IoT threats. Babar *et al.* [137] proposed a taxonomy of attacks based on the attacker goal, such as Physical Threat, Communication Threat, Identity Management, and more. Mayzaud *et al.* [138] instead proposed an extensive taxonomy of attack for RPL-based IoT networks, but focusing only on such routing protocol. In our proposed taxonomies, we aim at classifying attacks from the high-level perspective of the specific attack and communication pattern used, as well as finding relationships between the features of the monitored devices/networks and the potential security issues, since these classifications are more useful from the perspective of designing a comprehensive IDS for the IoT.

4.7 Summary

In this chapter, we proposed two taxonomies, for IoT attack patterns, and for the relationship between different network/device features and security threats. Then, we presented the design and implementation of Kalis – to the best of our knowledge the first comprehensive, self-adapting, knowledge-driven IDS for IoT, able to detect attacks in real time across a wide range of protocols and security goals. The evaluation

of our complete prototype shows that our knowledge-driven approach makes Kalis effective and efficient in detecting anomalies and attacks in IoT networks.

The scale of IoT devices and their exposure to the untrusted Internet make them desirable for enrollment in botnets. Therefore, in complementation of the monitoring performed by Kalis, we develop specific techniques to address this threat, presented in the next chapter. Moreover, in some cases, the detection can still bear some ambiguity with respect to the root cause of a security incident. However, an accurate diagnosis is crucial to an effective recovery, and we address such problem with the work presented in the upcoming chapters.

5 ROUTER-BASED DEFENSE AGAINST IOT-BASED BOTNETS

While monitoring the local sensor and IoT network through our IDS technique is critical, such systems are exposed from threats coming from the untrusted Internet as well. It is thus necessary to extend the monitoring also to the interactions that such devices have with remote services. One of the biggest threats that the rapid diffusion of IoT can enable, is that of botnets. The already mentioned large number of devices, together with their insecurity and surprising computational power, make IoT devices perfect tools to carry out powerful distributed denial of service (DDoS) attacks. While the actual effectiveness of IoT devices in carrying out such attacks has not yet been investigated, recent news show that attackers already have understood and exploited their potential in real-world attacks [139].

Current defenses are not ready to counter such novel botnets. Techniques that have proven effective against previous "conventional" botnets can be grouped in two main categories: *honey pots* and *anomaly detection*. The former category includes approaches that aim at purposefully exposing a vulnerable machine, hoping to get it infected as part of the botnet, with the goal of infiltrating the botnet and take it down. However, for botnets organized according to a peer-to-peer (P2P) structure, such defenses are not very effective. In fact, the infiltration into a portion of the network only results in the removal of a few attacking machines [140]. While extracting information about the attacker's network from a honey pot machine can be easily achieved, for IoT devices it is rarely possible to access such information after the infection, due to the lack, or extreme limitation, of user interfaces. Moreover, this defense technique is reactionary, as it aims to shut a botnet down once it has exposed itself, and cannot prevent an attack from taking place or the botnet from spreading. The second category of defense mechanisms, *anomaly detection*, includes approaches that build a model of "normal" behavior for an endpoint – usually by means of

statistical techniques – and then leverage such model to detect outliers that could reveal undergoing attacks. Such approaches, however, are very complex to design and tune, as they are meant to target general purpose machines whose activities are hardly profilable in a complete manner, thus subjected to a large amount of false positives. These considerations make it evident that previous defense techniques are hardly applicable to this new threat, and a design for an effective defense is thus needed.

In general, it is easy to understand that developing a "blanket defense" for IoT devices against botnet infections is an unachievable goal. In fact, the fragmentation in hardware platforms, operating systems, architectures, and proprietary firmware make it impossible to develop a one-size-fits-all defense strategy. Moreover, even a single vulnerability in one of these devices may easily vanish the entire defense work: an attacker able to exploit that single vulnerability may be able to replace the firmware of the device with malicious software, potentially spreading the infection to other local IoT devices. For these reasons, a more effective defense strategy is to leverage the connecting point of all the IoT devices, that is, the router that connects them to the Internet.

In the work presented in this chapter, we focus on both attack and defense aspects of IoT botnets. First, we thoroughly investigate the attacking potential of a botnet composed by IoT devices and construct a taxonomy based on our findings. From the defense point of view, we address the shortcomings of existing mitigation techniques and use such experience to tailor defense mechanisms to the peculiarities of IoT devices. Specifically we propose Heimdall, a lightweight, whitelist-based policy mechanism with dynamic profile learning capabilities designed for use on routers. We implement a complete prototype and evaluate it, showing that uncompromised devices see no disruption to their functionality while malicious traffic from compromised devices is immediately blocked, all with negligible overhead.

To summarize, the contributions of our work are the following:

- The evaluation of the attack potential of hardware devices and construction of a taxonomy of IoT devices;
- The design, implementation and evaluation of a centralized, lightweight defense mechanism able to prevent IoT devices from being used as part of botnets, while not disrupting their normal day-to-day functionality.

5.1 Background

In this section, we discuss the threat model that we address, a brief overview of botnets, and a summary of the platform used.

5.1.1 Threat Model

We consider a standard network comprised of a router with both wireless and hardwired connections to various IoT devices. Within such network, there may be devices whom have a base station and have their own communication protocol to their sub devices. An example of this is the Arlo home security system, from which a single base station can connect to multiple wireless home security cameras. However, these wireless cameras are not network bound to the router and invisible to the network. Within this model, there are no devices that communicate among themselves while also being bound to the router. This constraint is reinforced by the fact that there currently is no established general communication protocol among IoT devices; instead, each one of the device platforms interacts with each other via their public cloud APIs. Thus, if an attacker wanted to attack a device, they would be required to communicate with the victim through the gateway router. We assume the traditional network communication pattern in which a device willing to communicate to a destination (e.g., a remote service) first resolves a domain name to an IP address by means of a DNS query, and subsequently uses such IP address to communicate. This pattern is common to IoT devices as well as traditional computing systems.

5.1.2 Botnets

A "botnet" is a network of compromised machines (bots) running malicious software under a command and control (C&C) infrastructure. Usually, the controller of the botnet compromises a series of systems using various techniques – such as Trojan horses, worms, and viruses – to install a bot enabling remote control of the target machines. These "zombie" machines are then remotely controlled by the attacker (botmaster). Botnets with a large number of compromised machines have enormous bandwidth and computing capability. Such networks are utilized by botmasters for initiating various malicious activities, such as email spam, DDoS attacks, password cracking, key logging, and, most recently, crypto currency mining [10]. Modern bots can even automatically scan whole network ranges and propagate themselves using known vulnerabilities and weak passwords on other machines. After a successful invasion, a bot may use Trivial File Transfer Protocol (TFTP), File Transfer Protocol (FTP), or HyperText Transfer Protocol (HTTP) to transfer itself to the new compromised host. The binary is then executed and tries to connect to the botnet. Today, there are primarily two organizations to botnets, both with their own unique advantages.

Centralized Botnet

In this approach, the botmaster distributes a command over the botnet via multiple C&C servers in order to hide their own identity. Figure 5.1 shows the basic control communication architecture of a typical C&C botnet (in reality, a centralized botnet would have more than two C&C servers). Through the use of multiple C&C servers, centralized botnets are difficult to shut down. This architecture is easy to construct and efficient in distributing botmaster's commands with low latency. However, it has its major weakness in the C&C servers, as they are a single point of failure; shutting down these servers would cause all the bots to lose connection with their botmaster.

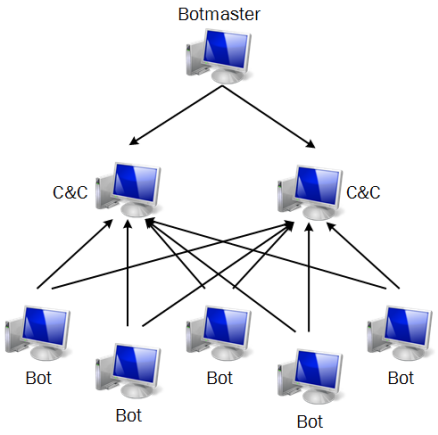


Figure 5.1. Example architecture of a C&C botnet

P2P Botnet

P2P botnets are an architecture that evolved from the centralized botnet architecture. As discussed in the previous paragraph, centralized botnets depended upon C&C servers for the botmaster to issue commands to all of the bots, making it very easy for defenders to monitor and shutdown these servers by creating a decoy to join a specified C&C channel. Thus, a natural strategy is to utilize a P2P control mechanism to circumvent defenses that aim at shutting down the C&C servers. Just like P2P networks, which are resilient to dynamic churn (i.e., peers join and leave the system at high rates) [141, 142], P2P botnet communication will not be disrupted even when losing several bots. As seen in Figure 5.2, in a P2P botnet there is no central server; bots are connected to each other, and act as both C&C server and client. P2P botnets have shown advantages over traditional centralized botnets. As the next generation of botnets, they are more robust and difficult to defend against.

5.1.3 Defense Platform

For the development, deployment and testing of our defense mechanism, we use the Linksys WRT1900AC router [143] running OpenWRT Chaos Calmer [144]. The

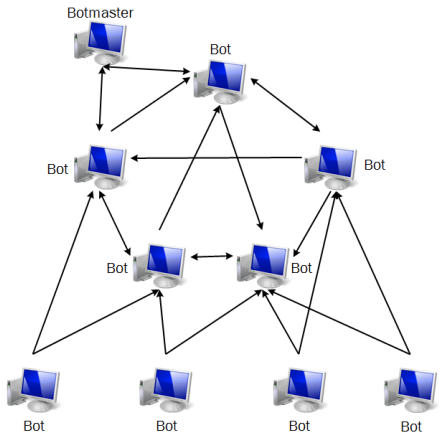


Figure 5.2. Example architecture of a P2P botnet

Linksys WRT line of routers is a popular network solution used in various applications, and it is a prime example of routers that we would find as the gateway between IoT devices and the Internet. We decided to use the OpenWRT firmware over stock firmware, as OpenWRT allowed us to easily make modifications to the router. OpenWRT is a popular open source Linux distribution, primarily used on embedded devices to route network traffic. OpenWRT is optimized for size and efficiency, so to be small enough to fit into the limited storage and memory available in home routers, while at the same time supporting a wide range of features with its 3500 optional packages [145]. Because of its many advantages, OpenWRT is used as the basis of production firmware in consumer routers, such as those found on Asus routers [146].

5.1.4 Virus Total

In addition to the utilization of the Linksys WRT1900AC router in the development of the defense, we utilized a 3rd party security information aggregation service called VirusTotal [147]. VirusTotal is a free online service that analyzes files and URLs, enabling the identification of virus, worms, trojans, and many other forms of malicious content detected by antivirus and scanners. VirusTotal acts as an informa-

tion aggregator, with data coming from over 60 different antivirus engines, website scanners, file and URL analysis tools [148]. It is therefore an excellent validation endpoint that Heimdall leverages in monitoring.

5.2 Related Work

The work presented in this chapter touches on several different research areas.

Among the defense techniques against botnets, two directions are particularly interesting: the use of statistical analysis anomaly detection techniques – aimed at blocking the attack from leafs to victims – and the infiltration in the structure of the botnet through honeypots – aimed at identifying the root to kill the botnet.

In the anomaly detection area, a great number of big data/statistical approaches have been proposed. Many of them focus on exploiting data mining techniques to detect outliers. For the case of IoT, such techniques require special care as their effectiveness rely entirely on the completeness and correctness of the "normal" profile. In Heimdall, we generate independent profiles for each IoT device by leveraging the correct Internet destinations needed for their functioning, and enforce those profiles with different levels of strictness for best effectiveness.

In the scenarios in which autonomous devices are able to directly talk to each other, Murynets *et al.* [32] investigated techniques for clustering devices based on their communications. Therefore, the profiles built show normal, expected interactions among devices, and use those to develop anomaly detection techniques for cellular Machine-to-Machine (M2M) communications. In IoT application scenarios, the manufacturers have in their best interest to use cloud services as intermediaries for all device-to-device communications; therefore, the lack of direct communication, while making the M2M-communication-based defenses not applicable, can actually be used to our advantage in the design of the defense technique.

Liu *et al.* [33] proposed an anomaly detection model for IoT based on an artificial immune system, which uses agents deployed at several gateways to collect statisti-

cal data and share those with a central service. In a related solution, Gonzales [34] analyzes how artificial immune systems can benefit anomaly detection; various representation schemes are investigated, and four different algorithms for the generation of detectors are proposed.

One first interesting work about measuring and mitigating P2P-based botnets is by Holz *et al.* [149]. They used the Storm worm as case study, and identified a three-phase process for botnet response: first, the extraction of relevant botnet bootstrapping information from a captured bot (such as the IP addresses of initial peers, and more); second, the infiltration of the botnet to monitor it from the inside; third, exploiting the publish-subscribe communication to take over the botnet and thus mitigate the attack. The authors correctly discuss how P2P botnets are technically hard to tackle, supporting our claim about the criticality of our motivations.

5.3 Attack

In this section, we discuss the challenges and advantages of IoT in terms of attack and botnet design.

5.3.1 Challenges in IoT Attack Design

Despite the fact that embedded devices have grown significantly in capabilities over the recent years, they still have many limitations. IoT devices suffer from the same limitations, which creates unique challenges in the construction of IoT botnets. In our analysis concerning how to construct a IoT botnet we identified a number of critical challenges unique to IoT devices that we discuss in what follows.

Low Resources

As with all embedded systems, IoT devices have very limited resources available for operation, including CPU cycles, memory, and available network bandwidth. Such

limitation is not an issue with respect their expected use, as IoT devices generally do not generate much traffic and do not require massive computations to fulfill their intended purpose. However, in a botnet that is not the case. Depending upon the intended use of the botnet, limitations of specific resources are a major issue. For example, limitations in the CPU cycles would be of primary concern if the purpose of the botnet were to mine crypto currencies, whereas network bandwidth limitations would be the main concern if the purpose were a DDoS attack.

Variety of Devices

IoT provides countless device, each specialized on different tasks. Each such device has its own hardware platform and software implementation. This leads to many different architectures and chipsets that need to be accounted for when crafting the malware and attempting to hijack devices during the botnet construction. Such variety vastly increases the difficulty in creating botnets, as no longer are attackers able to focus on a single architecture with a single operating system. However, as discussed next, this is also an advantage to the botnet, as the variety of devices increases the difficulty in shutting it down.

5.3.2 Advantages in IoT Attack Design

While there are many disadvantages from utilizing IoT devices in the construction of botnets, there are also many advantages, as we highlight below.

Variety & Number of Vulnerable Devices

It is estimated there will be over 20.8 billion IoT devices online by the year 2020, while there already are about 4.9 billion connected devices in 2015 [150]. This means that there will soon be an even larger number of devices for botmasters to target. Previously, when developing defenses against botnets, a single architecture, namely

x86, was considered. This allowed researchers and vendors to deeply analyze the specific strain of malware and how it worked in compromising its targets [151], to then develop a blanket defense against it. However, with such large numbers of devices and services, there is a much larger variety of devices to be targeted. This allows future botmasters to construct a botnet with a high entropy in device types, enabling them to construct a very robust network. Now, when a vendor patches a vulnerability utilized by the botnet thus removing the botmasters malware, it will only affect a small portion of the botnet, leaving the rest of the network unaffected.

Ease of exploitation

As the OWASP 2015 report highlights, IoT security is the worst of all domains. The reason is that IoT is a conglomerate of technology that links network, application, mobile, and cloud technologies together into a single ecosystem [152]. Such heterogeneity introduces multiple failure points all within a single application, making it an ideal target for attackers. A report by HP describes vulnerabilities that have been exploited, such as credentials being sent over clear text, network ports listening with root shells without a password, private data leakage, and a range of web and mobile vulnerabilities [14]. A second study by HP evaluated IoT home security systems and found the following: 100% of systems were vulnerable to account harvesting via the cloud interface allowing attackers to brute force credentials; 100% allowed weak password, such as 123456; 100% failed to implement account lockout defense; 90% lacked a two-factor authentication option; 70% had security posture variance between their cloud, web, and mobile interface; 70% had egregious issues with their software update systems; 50% exhibited improperly implemented SSL/TLS encryption. In such study, it was noted that one of the systems analyzed went as far as retrieving firmware via FTP, allowing the capture of credentials that would give an attacker write access to the update server [153]. This would be a botmaster's dream case, allowing them to quickly and effortlessly distribute their malware to every device utilizing that server

at once. As the list of issues found in these devices increases, it is trivial to conclude that targeting IoT devices will be a unavoidable step in the evolution of botnets.

Diurnal Dynamics

As Dagon *et al.* pointed out [154], the online population of traditional botnets has a clear "diurnal" dynamic, due to many users shutting down their computers at night. In a specific time zone, the peak online population of a botnet could be as much as four times the valley level online population. However, within the IoT this is not the case. By definition, IoT devices are constantly online, which creates a diurnal dynamic very helpful for a botmaster. Thus, when leveraging IoT devices, a botmaster will no longer be restricted to certain time periods to launch optimal DDoS attacks, or achieve optimal spread of new malware.

Device Taxonomy

As discussed previously, many different platforms and architectures enable IoT. Out of those, we selected multiple exemplary devices from popular categories of consumer IoT devices and classified them based on processor power and specificity of the purpose they serve. Figure 5.3 shows the result of our classification, using a scale from 0 to 100 for the varying degree of the measured metrics. We observe a very clear pattern among the devices as the specificity of the devices increases. Low specificity devices such as an Odroid or Raspberry Pi, which are examples of single board computers, run a full Linux distribution on their respective platform, allowing for users to rapidly develop various applications to run on them. These boards are perfect examples of the hardware that would be found in real-world deployments. As we move along the specificity scale, we see another grouping of low power platforms such as Arduino, Photon, and Galileo boards. These are all examples of development hardware which allows rapid prototyping of their respective IoT devices, while being more limited. Following, the next grouping is of IoT devices that have their own

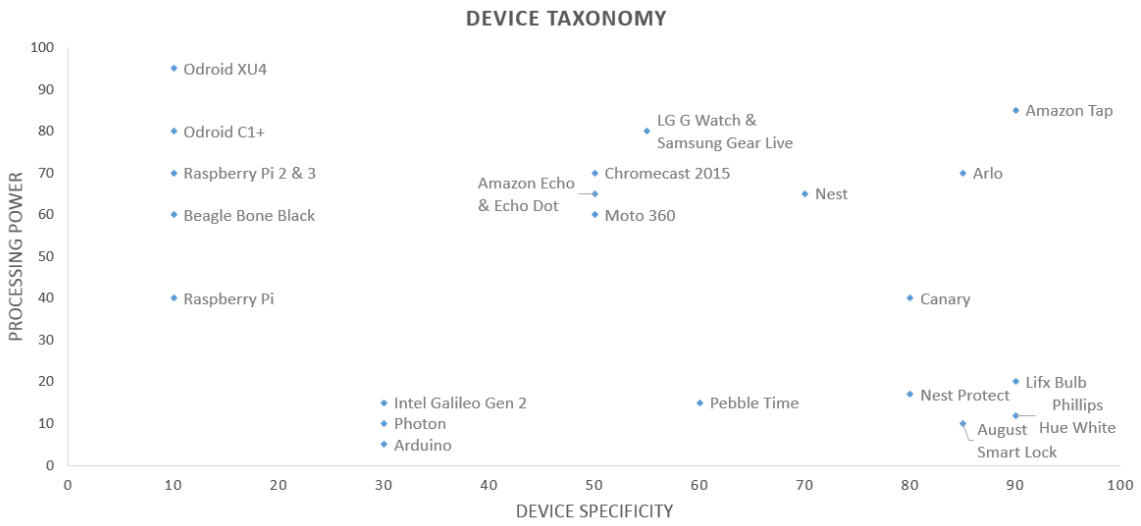


Figure 5.3. Device taxonomy of surveyed IoT devices

native app support, such as Amazon Echo, Moto 360, and the Samsung Family Hub Fridge. These devices are examples of production IoT devices that not only serve their specific purpose but also allow for additional user configurations and installation of applications. Last, we see the cluster of very specific devices that have no built-in interface and are all interacted with via their respective apps. Devices within this category are the most attractive to attackers, due to the fact that there is limited user interaction with them but they have a wide range of processing power.

5.4 Heimdall Defense Technique

In this section, we discuss the challenges and advantages of IoT in terms of defense technique design, and describe Heimdall, our proposed defense techniques.

5.4.1 Challenges

Designing a defense technique against IoT botnets is challenging. Our anomaly detection-based approach, Heimdall, is based on a strategy that builds a profile of the

normal behavior of each device, and enforces a white-listing policy that only permits actions compliant with such profile.

In the specific case of IoT devices, an appropriate profile should include the destination servers that the device contacts under normal operation. However, the first obstacle that becomes apparent by analyzing the traffic patterns on common consumer devices is that their backend services are usually hosted on load-balanced, public cloud infrastructures. Consequently, the IP address for a correct destination server is bound to change frequently, and therefore an IP-based whitelisting approach for the profiling and policy enforcement would result in disrupting the normal functioning of the devices.

5.4.2 Advantages

Despite the challenges of designing defenses against IoT botnets, a careful consideration of the specific behaviors of Io"T devices highlights specific features of IoT devices that can be leveraged to design a lightweight yet effective defense.

Building a profile for anomaly detection for general purpose machines – such as personal computers or workstations – is very hard because of all the different behavioral patterns that they can present. Conversely, IoT devices usually perform a well-defined, small range of operations. Thus, within Heimdall, such recurring, static set of possible actions can be leveraged when determining the normal behavior to be included in the profile. It is also important to notice that anomaly detection techniques for IoT do not need complex statistical or probabilistic inference models for building profiles and enforcing whitelist-based policies. Instead, a small whitelist can be used in combination with a traffic validation service to obtain a very effective and more accurate defense mechanism.

Moreover, the fact that the destination IP addresses for the devices' network communications often change for load-balancing actually leads to an advantage. In fact, by monitoring the domain name system (DNS) queries issued by the devices,

one can obtain a human-readable set of valid server destinations to whitelist. This opens the way for interesting scenarios. In fact, since this set of valid hostnames is consistent across all the devices of the same kind (e.g. all the Nest thermostats will reach out to `frontdoor.nest.com`), it would be possible to ship Heimdall to the end users already including some partially-built profiles.

Lastly, when placing the defense mechanism on the gateway router that all IoT devices in the local network use to reach the Internet, there is the possibility of missing potential device-to-device direct communications. In fact, some devices are able to cooperate and exchange state and commands, as in the case of the August Smart Lock and Nest integration. However, by analyzing the design choices of the vast majority of consumer IoT devices (and of all the devices in our evaluation testbed), any communication between devices goes through cloud API services, and never directly from a local device to another. This is a very reasonable and effective design choice by the manufacturers, as it centralizes the communication specifications and enables extensible interoperability, while at the same time reducing the attack surface.

5.4.3 Heimdall Architecture

Our approach is implemented as an event-driven defense algorithm – i.e., it is triggered on-demand by external events generated by the IoT devices, such as a new device joining the local network or the request by a device to send a packet to the Internet – in order to improve performance and react to external triggers. We now discuss the event-driven defense algorithm. The pseudocode for such algorithm is shown in Algorithm 3. Note that `dest d` in the algorithm can be both a domain name or an IP address, depending on the packet currently being analyzed (i.e., whether it is a DNS query or a subsequent packet destined to the resolved IP address.)

Whitelist Manager. While Heimdall is active, the whitelist manager is in charge of maintaining each connected IoT device's whitelist. Upon detecting that a new device is connected, a list is created for said device and populated with the device

ALGORITHM 3: Overall event-driven Heimdall pseudocode

```
Function(WhitelistMgr.onTraffic (device D, dest d)) create empty WL_D if not exists;
if d not in WL_D then
    if realTimeValidation then
        if d not in DestCache then
            DestCache.checkDest(d);
        if d malicious in DestCache then
            return REJECT;
    add d to WL_D;
return APPROVE;
Function(TrafficMgr.onTraffic (request req, device D, dest d)) if d in Blacklist then
    drop(D, d);
    return;
r = WhitelistManager.onTraffic(D, d);
if r == REJECT then
    drop(D, d);
    return;
let req from D go to d;
intercept response R (don't let go to D yet);
if req is DNSQuery then
    if r.IP != null then
        if r.IP != R.IP then
            r' = WhitelistManager.onTraffic(D, R.IP);
            if !r' then
                if !selfCorrection then
                    drop(D, d);
                    return;
                rewriteReply(R, r.IP);
    WhitelistMgr.associateForSession(r.IP, WL_D.d);
release R;
Function(TrafficMgr.drop (device D, dest d)) send(D, "Destination Unreachable");
sendAlert(D, d);
Function(Auditor.audit ()) forall the entry in DestCache do
    DestCache.checkDest(d);
Function(DestCache.checkDest (dest d)) r = queryVirusTotal(d);
if r benign then
    remove from DestCache.malicious[];
    add to DestCache.benign[];
else
    if d in DestCache.benign[] then
        remove from DestCache.benign[];
        WhitelistManager.purgeFromAllWLs(d);
        sendAlert(d);
    add to DestCache.malicious[];
return r;
```

destinations as they are enumerated. The primary purpose of this is that, once a destination is validated, any future validation is streamlined to minimize traffic delays. While monitoring the communication of each device, the whitelist manager is also auditing all whitelisted domains. Carrying out this continuous verification, Heimdall

can ensure that even if a destination within a whitelist becomes malicious after the initial verification, it will be removed from all devices' whitelists and added to the global blacklist within a short time period. This will block all future communication to or from that destination, such that, even if the device is compromised, it would not be able to participate in the botnet. With the combination of validation at enumeration time and continuous auditing of the whitelists, all entries in a devices profile will then constitute all and only the legitimate destinations for the device's traffic.

Traffic Manager. During the execution of Heimdall, the traffic manager analyzes each packet sent and its destination. This component is responsible for leveraging the whitelist manager and validating that each destination for a specific device's communication is legitimate. Upon each new communication, the traffic manager checks to see if the destination is contained within the global blacklist or the device's whitelist. If the destination is within the blacklist, then the communication is instantly dropped, otherwise it is allowed through. However, if the communication is a DNS request, the traffic manager has the secondary duty of validating the DNS response as well. Upon receiving the response from the DNS server for a specified request, the traffic manager compares the DNS response against the report returned from VirusTotal. If it is found that the destination IP addresses do not match, then there is a high probability that a DNS poisoning attack is occurring against that device. The response packet is then rewritten to contain the correct IP address from VirusTotal and then sent to the device, preventing attackers from externally forcing a device to communicate with a malicious destination whose domain was previously whitelisted.

Multitiered Policy Enforcement. Since each IoT device will have a different set of requirements for its functionalities, Heimdall supports two different modes of validation, namely maximum throughput and real-time validation. The maximum throughput mode is a simplified validation where each destination is checked against the global blacklist and if not found there automatically each destination is added to the whitelist and then audited at a later time by the whitelist manager. This mode

incurs a constant overhead for each communication as at no point are transmissions being held for validation before they are forwarded. On the other hand, the real-time validation mode looks at every single communication and, if the destination is not in the device's whitelist, it is validated with VirusTotal. This is the strictest model, since a domain known to be malicious would never be added to a whitelist. Under both policy modes, however, the DNS requests are always validated, as they are the entry point for all communications for any device.

5.4.4 Implementation Details

We implemented a prototype of Heimdall operating on the Linksys WRT1900AC router. As highlighted in Section 5.1.3, this platform was chosen due to its support from the OpenWRT project. With OpenWRT we were able to add new features to the router by installing our defense tool at the gateway for connected IoT devices.

When a new device is connected to the router, Heimdall immediately isolates it and begins analyzing the devices traffic. In order to implement this, we utilize the IPTables [155] utility in combination with our custom proxy and the VirusTotal service. We utilize IPTables to forward all traffic for each of the IoT devices through our proxy running on the router. The proxy is thus able to analyze each outgoing communication and DNS response, to validate the destinations accordingly by utilizing VirusTotal services. Based on the input from the proxy, we query VirusTotal upon encountering a new destination. By leveraging the 63 data points including known trusted services such as BitDefender, Kaspersky, and Malwarebytes in the report VirusTotal provides [148], Heimdall is able to judge whether the destination is known to be malicious. If it is found that the destination is malicious, the domain and IP address are added to the blacklist to prevent all further communication on the network to that destination.

5.5 Evaluation

In the evaluation of our defense, we utilize five very popular IoT devices, namely Nest Thermostat [156], August Smart Lock [157] with August Connect [158], Amazon Dash Button [159], Arlo Home Security System [160], and Lifx smart bulb [161]. These devices were selected due to their features including a mobile application, user account, cloud-based back-end, automated features, and inter-device integration. Our evaluation consists of three parts. First, validating that there was no interrupted service while Heimdall is active. Second, validating that the device could not reach a known malicious destination. Finally, we evaluated the network overhead of Heimdall when active on the router.

Attack Evaluation

Since it is not in the scope of this work to construct a botnet, we simulated common attacks to benchmark the capabilities of the different devices we were using. We utilized the Hyenae Network Packet Generator Tool [162] to do this. Hyenae is a flexible, platform-independent network packet generator. It allows users to reproduce multiple DoS and DDoS scenarios such as TCP-SYN and ICMP-Echo flooding. For the purpose of our experiments, we tested three distinct attack scenarios, namely *TCP-SYN*, *IMCP-Echo*, and *UDP flooding*. For each attack, we limited it to one million packets and monitored how long the execution took, to calculate the average Mbps each device was capable of. In order to evaluate the capabilities of the Linux boards in a botnet attack scenario, we implemented a hybrid P2P botnet with our boards and benchmarked their capabilities. To do this, we utilized the network benchmark tool *iperf*, a platform-independent tool for performing network throughput measurements [163]. For the purpose of our experiments, we tested both *TCP* and *UDP* throughput between each board and the test server. For each result we took the average of twenty-five benchmarks.

The Desktop averaged at more then 930 Mbps for *TCP* and over 800 Mbps for *UDP*. This is due to the fact that modern machines are equipped with gigabit ethernet connections that allow for very fast network connectivity, in addition to the south bridge on the motherboard that offloads a majority of the computational requirements for network communication. The next highest performing device was the Odroid xu4. This board averaged $\tilde{9}19$ Mbps for *TCP* communication and $\tilde{5}95$ Mbps of *UDP* traffic. This is very impressive due to the fact that the Odroid xu4's network connectivity is comparable to that of a full fledge x86 based desktop. Due to this an attacker could easily replace a desktop with an IoT device utilizing similar hardware such as the Odroid xu4. The next highest performing device was the Odroid c1. This board averaged greater than 400 Mbps of both *TCP* and *UDP* communication. It is interesting to note that this board – despite having limited processing and memory capabilities in comparison to the Odroid Xu3 – still out performed it. This is due to the fact that the Odroid C1 has a dedicated network controller, the Realtek RTL8211F [164], whereas the Odroid Xu3 utilizes the LAN9514, a shared LAN/USB controller [165]. The third highest performing board was the Odroid Xu3 with optional USB 3.0 gigabit adapter, averaging more than 405 Mbps for *UDP* traffic and 220 Mbps for *TCP* traffic. The fourth and fifth highest performing boards are the Raspberry Pi 2 and Raspberry Pi, which maximized their fast ethernet connections. Despite the deeply different range of resources, each one of these boards is actually bottlenecked by the network hardware implementation. As these chips improve, we can expect to see more lower-end IoT devices increase to more closely match that of full x86 machines.

Based upon our experimental results we can now estimate how many IoT devices would be required to build a botnet that would compete with its traditional x86 based counterpart. Assuming that there is a botnet of size 100,000 x86 devices and with a 55% peak of its population online at maximum due to the diurnal dynamics of the botnet. This botnet would achieve a peak of 49.5 Tbps, given that each device can average 900 Mbps. To achieve this within an IoT domain we would then need 90,000

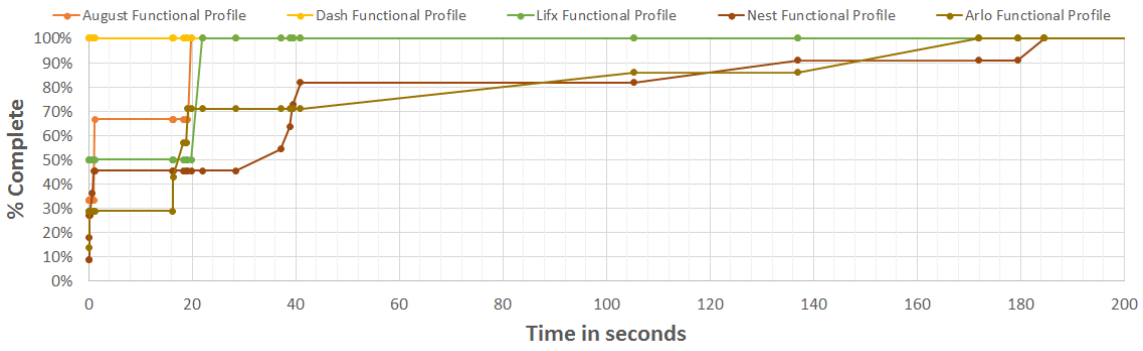


Figure 5.4. Functional profile completeness

IoT devices, assuming an average of 550 Mbps from each device. This may appear as a massive number of devices but in comparison to the 20.8 billion new devices expected to be online by 2020 it is only 0.00000432% of the population. In addition in a domestic scenario where the internet connection of the devices is limited by the provider it is worth noting that even the lowest performing board, the Raspberry Pi, would match that of its x86 counterpart. Thus despite the limitations in the network capabilities of these embedded devices a botmaster could easily construct a very effective effective botnet.

Validation of Profile Health

We validated how quickly consumer devices would expose a complete enough behavior in terms of destinations reached. We went through the initial configuration of each device within our lab, following the instructions provided to connect our accounts to the respective devices and sync them with their cloud services. We captured snapshots of the whitelists built by Heimdall – with no ongoing attacks – at 1 hour, 24 hours, and 1 week of normal execution for all devices. We define *nominal completeness* of a device's whitelist as the completeness w.r.t. the entire set of possible destinations that such device will ever attempt to communicate with. Conversely,

we define *functional completeness* of a device's whitelist as the completeness w.r.t. to the minimum set of destinations that need to be reachable for a device to carry out its normal functionality. A functionally complete whitelist is a subset of the related nominally complete whitelist. This distinction is necessary because IoT devices make heavy use of load balancing techniques in reaching their cloud servers. For this reason, some devices might try to contact, at different times, different domain names corresponding to various replicas of the same service. However, if traffic towards one of those replica domain names is blocked, it is often the case that a device will simply attempt another one as fallback, having therefore no interruption in the regular functionality. Thereofre, while including all those replicas' domain names concurs to a better nominal completeness for a whitelist, the functional completeness refers to collecting the minimum set of destinations that allows an uninterrupted functioning of the device. We used the 1 hour, 24 hour, and 1 week traffic snapshots to evaluate the nominal completeness and functional completeness of the whitelists at those points in time. Throughout the whole week of testing, we observed that Heimdall never interfered with legitimate execution, not preventing any correct activity. Upon analyzing the network traffic during the experiment, we found that not all the legitimate destinations were observed during the first hour, leading to whitelists that were initially not 100% complete until later on, as seen in Figure 5.5. This is due to the fact that some devices such as Arlo only attempted to enumerate other domains used in load balancing after a few hours. For example, in the first hour Arlo enumerated two separate domains `vzweb07-prod.vz.netgear.com` and `vzweb06-prod.vz.netgear.com`. However over the course of the week it attempted to enumerate `vzweb01-prod.vz.netgear.com`, `vzweb04-prod.vz.netgear.com`, and `vzweb05-prod.vz.netgear.com`. Thanks to the dynamic learning of Heimdall, at no point in time was the device functionality compromised. Thus, as seen in Figure 5.4 the functional profile reaches 100% completion in just over 3 mins. This means that after 3 mins of the device communicating, Heimdall has learned all required domains for the device to perform its tasks.

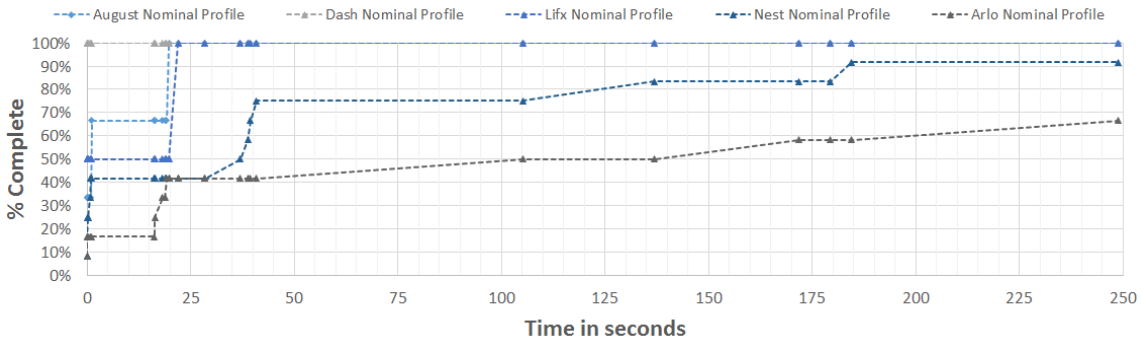


Figure 5.5. Nominal profile completeness

Validation of Blacklist construction

To test the enforcement phase, we used a Raspberry Pi 2 as a malicious device to impersonate each of the five consumer IoT devices after the learning phase was complete. In order to accomplish this we powered off the original device, disconnected it from the network, and spoofed the Raspberry Pi 2 MAC address to mimic that of the device. Upon connecting the Raspberry Pi 2 to the router, we proceeded to have it ping multiple domains that we knew were not on the whitelist, which resulted in those packets being blocked by the router. Next, we attempted to subvert the whitelist by pinging the target IP address directly instead of using their respected domain name. These attempts were again blocked as the IP addresses we targeted were not included in the whitelist.

Heimdall Overhead Evaluation

Lastly, we validated the overhead of Heimdall in terms of latency introduced in the network traffic. To benchmark it, we utilized the same Raspberry Pi 2. During this time, we used the network `ping` utility to ping an allowed domain from each of the devices. As seen in Figure 5.6, our results show less than 1% in overhead over 50 pings with Heimdall active and without Heimdall active. Such close-to-ideal performance

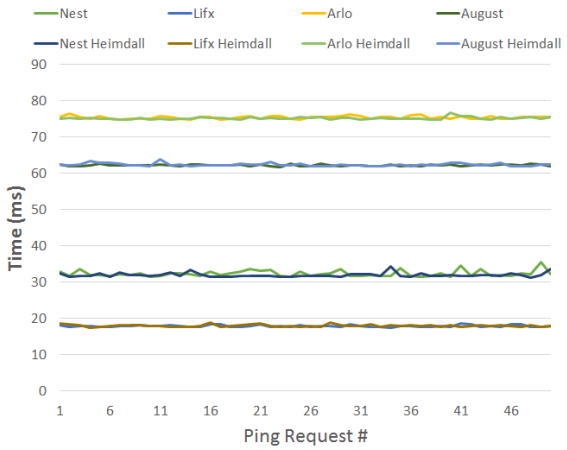


Figure 5.6. Heimdall latency

is due to the fact that Heimdall leverages the existing low-latency features of the iptables tool to enforce the profile, thus leveraging the existing optimizations and network functionalities.

5.6 Security Analysis

In this section, we provide a discussion of the security of our approach, and how a malicious attacker might try to bypass the defenses put in place by our work.

The goal of an attacker would be to bypass our whitelist-based defense mechanism. The scenarios in which this can be attempted are as follows.

An already-compromised device joins the network, and the attacker tries to exploit local device-to-device communication as a vehicle to spread the attack to other uncompromised devices on the local network. This is one of the most important scenarios, in which the attacker tries to get the IPs of the C&C servers to be whitelisted for other devices. As this is one of the most dangerous scenarios – given that the compromised device inside the local network represents conceptually an insider threat – we devoted special attention to it. There are two cases: (1) the malicious IPs are already blacklisted, or (2) the malicious destinations are seen by Heimdall for the first time. In

case (1), not only the compromised device will not be able to spread to other devices, as they will be blocked from reaching those destinations, but it also would not be able to reach such destinations itself. In case (2), at the first attempt to communication to malicious destinations, the verification carried out by Heimdall will lead to their blacklisting, then falling back on case (1). This covers all possibilities and ensures that no insider attack can manipulate the process and compromised the device or poison its whitelist.

The reliability of a destination changes over time. A destination initially allowed might be discovered as malicious later on. The auditing process carried out periodically by Heimdall takes care of refreshing all whitelists with the most up-to-date security statuses for all destinations. This limits the attack window to a short period of time, making it ineffective for DoS attacks.

A device's firmware receives an update. In this situation, the range of valid destinations for a device might change. Since Heimdall exercises a constant learning on new destinations and subsequent validation of them, after a firmware update the device will be able to attempt to contact new destinations and those (if valid) will be added to its whitelist. If the firmware update compromises the devices, no new malicious destinations can be reached anyway as they will be subject to the validation process as any other initial destinations.

The attacker carries out a DNS poisoning attack affecting the local devices. This could lead to a domain being verified as legitimate by Heimdall, but then being resolved to a false, malicious IP address. Since Heimdall performs an extra validation step to compare the publicly available DNS resolution with the DNS Reply from the local DSN server, DSN poisoning attacks will not be able to circumvent the system. In fact, the destination IP will eventually be blocked anyway before any actual communication can take place.

A malicious device (or development prototype) communicates with hardcoded IP address destinations, without DSN queries. In this case, Heimdall's destination verification process will still be able to verify the IP rather than the domain. The leveraged

information can include publicly available IP blacklists. If it is not yet known as malicious or benign, the destination can be temporarily allowed and the periodic auditing will block it as soon as the assessment changes, greatly limiting the attack possibilities even in face of unknown, hardcoded IP destinations.

Even though the whitelist for a device is correct (i.e., it does not contain the IPs of any botnet C&C server), the attacker tries to evade the detection. Unless the router is compromised, this scenario is once again covered completely by the whitelisting-based defense mechanism itself, and therefore the attacker's attempt will not succeed.

The attacker compromises the router. Even though it is out of the scope of this work, we also consider the scenario in which an attacker tampers with the router's firmware in an attempt to disable or alter Heimdall. This situation would require the attacker to have physical access to the router in order to either re-flash a new software image on it, or install or remove individual software modules installed on it. For the use cases in which this might be a real threat – such as unattended operation environments for routers in manufacturing or industrial scenarios – any remote software attestation technique available in literature would be able to mitigate this issue. In fact, the router's firmware could be digitally signed, and a cloud service could periodically perform attestation to verify that no tampering has occurred with the router's software.

From all the scenarios described above it is possible to see that Heimdall is thorough and effective.

5.7 Summary

In this chapter, we investigated the effectiveness of an IoT device based botnet and evaluated multiple Linux development boards to test their effectiveness in a DDoS attack. We showed that constructing such a botnet is quite easy due to the number of vulnerabilities identified in IoT devices. In addition, we proposed a defense technique, Heimdall, that mitigates these attacks by restricting the devices to their respective

legitimate domains. We validated Heimdall's approach and showed its effectiveness by testing it on several real-world IoT devices. Based on the results of our analysis and implementation of the defense, we argue that that whitelist-based anomaly detection is a practical and low-overhead defense against IoT botnets.

After the monitoring tools deployed in a network detect an attack, it is paramount for the system to quickly recover. However, an effective response relies on an accurate diagnosis of the root cause of the occurred security incident. We address such problem with the work presented in the next two chapters.

6 FINE-GRAINED ANALYSIS OF PACKET LOSSES IN WSNS

Once our monitoring techniques have detected an attack, determining its actual cause is crucial to an effective response action. In fact, a key requirement for highly secure network systems [166, 167] is represented by situational awareness (SA). SA typically refers to the gathering of a (possibly real-time) knowledge about relevant events happening in the network of interest. This enables an understanding of the impact of events and defensive actions on the network security, both immediately and in the near future. For wireless sensor networks (WSNs), packet loss is a class of events particularly relevant for SA, as it may result in relevant information to be lost, as well as undermine data quality solutions based on redundant data transmission [168]. However, just detecting the loss of data packets is not sufficient. Correctly diagnosing the causes of such losses is also crucial. Packet losses may be the result of either compromised or misbehaving nodes, or of attacks focused on network links. Determining the actual causes of packet loss attacks is vital in order to deploy effective responses to attacks as well as recovery and debugging actions.

Selective forwarding and *blackhole* attacks are examples of node related attacks, while interference is an example of a link related attack. Both these classes of attacks can result in partial or total packet loss. However, even thought the causes of such losses are different, current intrusion detection systems (IDSes) are typically only able to detect the packet losses, but are unable to determine the actual causes of the losses, whether node or link related [169–171]. Therefore, current IDSes need to be enhanced with techniques able to carry out a correct diagnosis of the cause of packet losses in the WSN of interest.

The work presented in this chapter addresses this need through the design and implementation of an approach for a *fine-grained analysis (FGA)* of packet losses to accurately diagnose their underlying causes. By analyzing all the links in the WSN

of interest, our approach is able to determine whether a packet loss attack is caused by a malicious node or a link-related problem. Our FGA tool builds profiles for the network links by leveraging packet resident parameters such as the received signal strength indicator (RSSI), the link quality indicator (LQI), and the packet reception rate (PRR). Upon the reporting of a packet loss by the IDS, these profiles support a thorough analysis, in order to determine the actual cause of the loss. Moreover, for packet losses caused by an interference on the links, our approach can estimate, with good accuracy, the location of the source of such interference. This knowledge empowers network administrators and automated incident response systems (IRSes) [43] to estimate the network regions affected by the interference – a crucial information, for example, in determining the affected nodes and letting the unaffected sensors take effective response actions, such as re-routing their data through links and nodes not affected by the interference.

The design of our FGA tool has many advantages that make it ideal for asynchronous systems such as WSNs. First, the analysis is event-driven and is carried out simultaneously at every investigating node. Second, multiple investigations can take place at the same time in cases when there is more than one misbehaving node in different network locations. Third, the investigation is carried out in a stealthy manner, giving no chances for the malicious node to interfere with the investigation results. Fourth, our approach is fully distributed, thus not relying on the base station (BS) to coordinate or perform the analysis. Instead, it is carried out solely by the direct neighbors of the faulty node or link. Lastly, our FGA tool has a low overhead, and can be implemented as a layer in many WSN systems. Sensor nodes have very limited computational and power resources. For this reason, to perform the required analyses, our approach leverages resident parameters – such as RSSI and LQI – that are available within every received packet. Therefore, unlike previous approaches [172,173], our FGA tool does not require any additional node or resource.

Our FGA tool has various applications, from forensics investigations to real-time response systems. As an example, forensic analysts may want to record the link mea-

surements for every packet in order to investigate and determine the nodes involved in some suspicious data transfers. Additionally, real-time IRSes may leverage the accurate analysis results of our FGA tool to enact better and more effective responses.

6.1 Adversarial Model

In this section, we introduce the assumptions that define the adversarial model.

We assume that the WSN consists of a large number of sensor nodes, and is fully connected via multi-hop communications. We assume the network topology to be designed so that every node has at least two disjoint paths to reach the BS. We also assume that an IDS is deployed at each node so that each sensor is capable of detecting packet loss attacks as well as data modifications [174–177].

The attacker has two ways of attacking the network:

- After the deployment, the nodes may be captured by the attacker, that will then be able to access all the information stored in those nodes, as well as reprogram them and control their actions. The attacker could therefore make node refusing to forward some of the packets (*Selective Forwarding* attack) or even all of them (*Blackhole* attack).

- The attacker may place a source of interference on the network surface, disrupting the wireless communication links between the nodes. The attacker can therefore cause the loss of some of the packets (*Low Interference* attack) or of most of them (*High Interference* attack).

Compromising a node to actively drop packets and introducing interference in the network are from a high level perspective, the two ways in which an attacker can disrupt network communications through a packet loss attack. For this reason, our adversarial model covers in fact many different attackers that aim at causing packet losses, since their means will eventually fall in one of the those two attack categories. Examples of other attacks that fall into such categories are the use of self-replicating

WSN worms that find vulnerabilities in the neighbors and propagate using packets as attack vectors (a node-related attack), or the alteration of environmental condition of a mote such as artificial exposure to very high or very low temperatures (a link-related attack).

Each node whose IDS detects a packet loss attack will investigate upon the loss; we assume the investigating nodes to be trustworthy and not to report false votes. This assumption is particularly important for the Majority Voting algorithm adopted as part of our approach. However, we will also present a variant of this algorithm able to relax this constraint, and thus able to tolerate up to a certain number of colluding investigating nodes.

We assume that multiple simultaneous attacks can be carried out at the same time in different parts of the network. In fact, one of the strengths of our approach is that multiple simultaneous investigations can be carried out.

6.2 Background

In this section, we introduce the basic parameters that are used in profiling the links among nodes in a WSN. As hardware platform, we use the $CC2420$ radio chips that are installed on the Telos nodes (see Section 6.7 for more details about the the $CC2420$ radio chip). The $CC2420$ chip provides two useful metrics: the RSSI and the LQI. It is important to note that our approach is not limited to the $CC2420$ radio chip, but can work on any radio providing those measurements, such as any newer radio based on the IEEE 802.15.4 standard.

The RSSI represents the signal power of the received packet and is measured in dBm. Its value is calculated over 8 symbol periods and stored in the $RSSI\_VAL$ register of the $CC2420$ radio chip. Chipcon, the manufacturer of these radio chips, has specified in the $CC2420$ datasheet that the signal power value is computed in dBm as $RSSI\_VAL + RSSI\_OFFSET$, where $RSSI\_OFFSET$ is about -45.

The RSSI value ranges between -50 and -100, with the higher value (less negative) representing a stronger signal.

The LQI is a measure of the current quality of the received signal and can be viewed as the chip error rate of the received signal. It is calculated over 8 bits following the start frame delimiter (SFD). According to the $CC2420$ specification, the measured LQI[1] is actually the average correlation of each symbol obtained by comparing the symbol that is supposed to be received and the symbol actually received (signal + noise). Its value usually ranges between 50 and 110, with the higher values representing better quality frames.

Finally, a third parameter used in our approach is the PRR, which is defined as the ratio of the number of successfully received packets over the number of packets sent between two neighbor nodes. A high PRR means a better link quality and a healthy communication link.

6.3 Network Profiling Management

In this section, we explain how we use the RSSI, LQI, and PRR parameters to profile each link between two neighbor nodes, and how we aggregate those individual profiles to determine each node's neighborhood profile.

6.3.1 Link Profiling

The purpose of link profiling is to provide a better understanding of the relationship between each node and its direct neighbors, which is a critical feature for determining the causes of packet losses. At initial network setup, the BS gets to know all the nodes by issuing a *HELLO* command and then requests each node, one at a time, to start the *Link Profiling* process. Each node, in turn, broadcasts M *dummy* messages, one every 10 milliseconds, to all its direct neighbors. The value of M must

[1]LQI values are generally calculated by a software converting their values to a range of $0 - 255$. The values are computed using the RSSI and the average correlation values. In this dissertation, we refer to the LQI as the average correlation, that is, the value that $CC2420$ chip denotes as LQI.

be chosen according to the stability of the network and its topology. The RSSI and LQI values can show high fluctuation in some networks; therefore a larger M will result in a slower initial profiling process, but more accurate profiles.

Since link profiling occurs during initial network setup, we assume that at this step there are no node or link related attacks, as such attacks can be easily detected by network administrators during initial testing. Figure 6.1 summarizes the steps in the link profiling.

When a node receives the *dummy* messages from its direct neighbor, it records the RSSI, LQI and PRR values for that specific link that connects the node to the sender. Afterwards, each node will create the specific profile of that link consisting of the averaged RSSI and LQI values, together with the PRR, to form the link profile triplet (AvgRRSI, AvgLQI, PRR) with each direct neighbor. Each link profile is saved locally at each node. Since link profiles depend on the parameters carried by *received* packets, our profiles are directional: the profile of the link between node n and node n' from the point of view of node n, denoted by $n \leftarrow n$', is different from the profile of the link from the point of view of node n', denoted as $n' \leftarrow n$.

If M is the number of *dummy* messages broadcasted by every node, and $R_{n \leftarrow n'}$ is the set of messages from node n' received by node n, then the components of the link profile $n \leftarrow n$' are computed as follows:

$$AvgRSSI_{n \leftarrow n'} = \frac{\sum_{i=1}^{M} RSSI\left(R^{i}_{n \leftarrow n'}\right)}{|R_{n \leftarrow n'}|}$$

$$AvgLQI_{n \leftarrow n'} = \frac{\sum_{i=1}^{M} LQI\left(R^{i}_{n \leftarrow n'}\right)}{|R_{n \leftarrow n'}|}$$

$$PRR_{n \leftarrow n'} = \frac{|R_{n \leftarrow n'}|}{M}$$

The corresponding link profile $n \leftarrow n$' is represented as the following triplet:

$$n \leftarrow n' = \langle AvgRSSI_{n \leftarrow n'}, AvgLQI_{n \leftarrow n'}, PRR_{n \leftarrow n'} \rangle$$

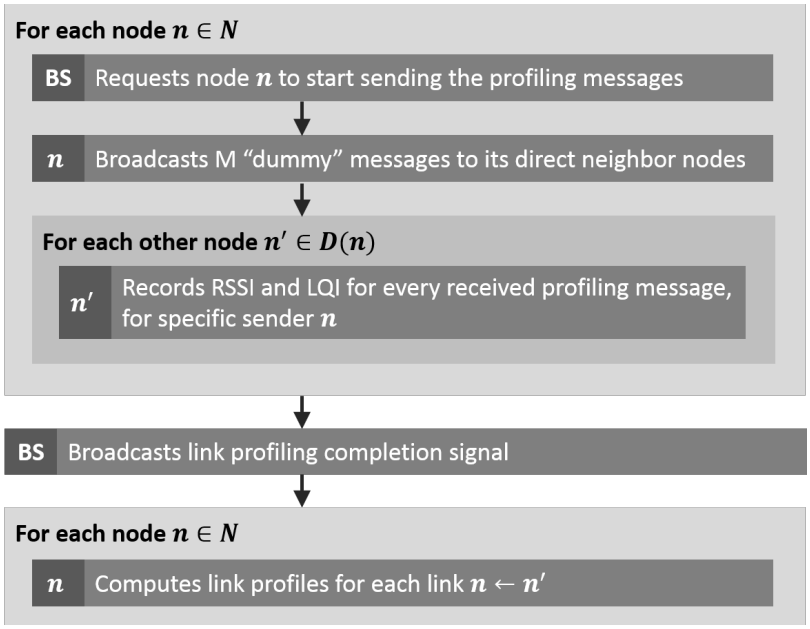


Figure 6.1. Profiling steps performed at initial network setup.

The storage required at each node depends on the number of direct neighbors of the node.

The initial *Link Profiling* process is vital for the FGA subsequent analyses. In fact, the RSSI and LQI measurements are piggybacked in the individual packets exchanged by the nodes, and the traffic load throughout the steady network might be highly different, providing more data for the profiles at some nodes and very little information at nodes where the traffic load is light. This would lead some of the averaged link profiles to show high variance and less reliability for their use in the analysis. Therefore, even though these profile will be updated over the lifetime of the WSN (see Section 6.3.3), the initial *Link Profiling* process guarantees a consistent base for the profiles to be built with comparable accuracy.

6.3.2 Neighborhood Profiling

We also define the profile of the neighborhood of each node by averaging all the link profiles of the direct neighbors of the node. The neighborhood profile is critical

in cases of strong interference for localizing the source of the interference and getting a better understanding on the areas of the network that are possibly affected, as discussed in Section 6.6.

Using the notation $n \leftrightarrow n'$ for a communication link between node n and n', we define the set of the direct neighbors of a node n as:

$$D(n) \subseteq N = \{n' \in N s.t. \exists n \leftrightarrow n'\}$$

By leveraging this definition, the components for the neighborhood profile of node n are computed as follows:

$$AvgRSSI_n = \frac{\sum_{n' \in D(n)} n \leftarrow n'_{RSSI}}{|D(n)|}$$
$$AvgLQI_n = \frac{\sum_{n' \in D(n)} n \leftarrow n'_{LQI}}{|D(n)|}$$
$$PRR_n = \frac{\sum_{n' \in D(n)} n \leftarrow n'_{PRR}}{|D(n)|}$$

and the corresponding neighborhood profile for node n, denoted $P(n)$, is represented as the following triplet:

$$P(n) = \langle AvgRSSI_n, AvgLQI_n, PRR_n \rangle$$

6.3.3 Profile Updates and Current Health Profile

During the normal lifetime of a node, its battery level will naturally lower. This phenomenon will result in communications showing a link profile (RSSI and LQI) for that node that is different from the one stored by all its direct neighbors. On the long run, as soon as the difference between the new "organic" profile and the original profile gets above the threshold for interference attacks, any packet drop will immediately be diagnosed as a link-related attack, even if it is not.

ALGORITHM 4: Algorithm for updating the current health profile for a generic link $n \leftarrow n'$

```
CHP[link] = initialProfile[link];
lastUpdate[link] = timestamp();
on snooping packet p from Node n' do
    Pp = calculateProfileForPacket(p);
    timeframe = timestamp() −lastUpdate[link];
    if abs(Pp − CHP[link])/timeframe) ≤ RATE then
        CHP[link] = aggregate(CHP[link], Pp);
    end
endon
```

In order to address this issue, the initial profile stored at a node A for one of its direct neighbors B is periodically updated based on overheard packets. Such updated profile represents the *current health profile (CHP)* of that neighbor B considering its normal lifetime variations. The investigation and profile comparison phases will then use this updated profile, denoted as $CHP[n \leftarrow n']$ as a reference to detect the cause of a packet dropping attack.

An important consideration in this case is how often the profiles must be updated, and which new values to consider for inclusion. Our algorithm only takes into account, for profile updates, those new values that can be considered as a natural variation due to battery decay. This constraint is necessary to prevent spurious values due to actual interference attacks or high fluctuations of the network from influencing the profiles and effectively "poisoning" them to carry out undetected attacks. A value is considered healthy if and only if the speed and smoothness of its change with respect to the current profile is gradual enough, i.e. is below a preset threshold rate. The WSN administrator needs to determine the threshold for the rate at which the change is organically gradual at startup, possibly with the help of automated tools. In fact, this is a parameter completely dependent on the used motes, the power source for each of them, and the power consumption of the application running on the nodes.

The algorithm for updating the current health profile for a generic link $n \leftarrow n'$ (denoted with "link" in the pseudocode) is shown in Algorithm 4.

At the beginning, the current health profile is set to the initial profile computed at network startup. After that, while the WSN is operating, the current health profile is periodically updated when a packet is snooped and its measurements are consistent with the organic decay rate specified (denoted with "RATE" in the pseudocode in Figure 4). The aggregation function simply incorporates the new value into the averaged profile components, but could also be customized to specify a weight for the new value into the CHP.

6.3.4 Adding, Removing, or Relocating Nodes

Our FGA tool is designed to perform the least possible changes in case of network modifications. As sensor nodes have limited processing power and storage, we have to ensure that our design can tolerate network changes, such as adding new nodes to or removing nodes from the network, as well as relocating existing nodes.

When a new node is added, the occurring changes only partially affect the direct neighbors of the introduced node. While the BS updates the network map, it requests the new node to build its link profiles with its neighbor nodes. The new node will also discover its direct neighbors and request them to profile their new common links. At this point, each neighbor node will add one more link profile to its existing list of profiles, and locally update its neighborhood profile as well.

In case a node has been removed, while the BS updates the network map, no further immediate changes are needed at the nodes. In fact, for some time the direct neighbors of the removed node will have one extra link profile recorded. Such extra link progile does not have any negative impact on future analyses, unless storage becomes an issue. As soon as a neighborhood re-profiling occurs during an investigation, each node will be able to determine which of the profiles it is storing are about links to nodes that have been removed from the network – as such profiles will not be updated during the re-profiling – and will therefore be able to remove from the memory such non useful profiles. Whenever storage is a major concern, it is also

possible for the BS to directly notify the neighbors of a removed node about this network change during its network map update. The neighbors will then proceed to immediately remove the profile of the removed node, and therefore reclaim storage right away.

The process of relocating a node capitalizes on the considerations already made for addition and removal of a node. Moving a node from a position in the network to another is decomposed by our system in two separate conceptual steps. First, the node is removed from the network, using the procedure described in the previous paragraph. Then, the node is added back to the network in its new position, following the procedure described in this section for adding a node. The need for these two separate steps stems from the fact that, since our profiling parameters (RSSI, LQI, PRR) are affected by location, re-positioning a node requires re-building its link and neighborhood profiles. As the BS updates its network map, the relocated node is introduced to new neighbors where neighborhood profiling is requested.

6.4 Diagnosis

In this section, we show the analysis steps to differentiate between different types of attacks that may cause packet losses. As introduced in Section 6.1, we assume each node to be equipped with an IDS to detect packet losses as well as data modifications [174–177]. In our work, the IDS will trigger the FGA only when the IDS detects that there is a packet drop attack that needs to be investigated upon.

Since our FGA approach focuses on differentiating between node-related and link-related attacks, the detection of the attack itself – versus a non-malicious packet loss – is delegated to the IDS. We believe that this approach provides our design with a more clean separation of concerns, and let the FGA module focus on the diagnosis of the attack, more than the detection of an attack versus a natural loss event. Either way, we believe that a false positive, due to the IDS improperly detecting a packet loss as an attack when it is not, will happen more rarely and with less persistence

than an active attacker dropping packets in the network by means of compromised nodes or interference. For these reasons, we think that a sporadic false positive due to a component out of the scope of the FGA such as the IDS, is way less detrimental than many false negatives to the overall security of the WSN.

Purpose of the analysis: The purpose of the FGA is to differentiate between the attacks that target the nodes from those that target the links. Through the detected attack, we can conclude whether the cause of packet drops is node- or link-related. For instance, *Selective Forwarding* attacks and *Blackhole* attacks are node-related attacks that cause partial and total packet losses, respectively. However, *Radio Interference*, or *Jamming*, attacks may also be responsible for packet losses as they have a negative effect on the network links and regions depending on the interference source location and strength[2]. Basically, the existence of interference can affect the RSSI and LQI values of received packets that passed through a noisy environment, and can sometimes impair the signal quality of other packets to the point that they become unreadable.

In the following subsections, we detail the various steps composing the analysis process carried out by the FGA tool upon the detection of a packet loss.

6.4.1 Analysis Startup and Evidence Collection

Our analysis aims at understanding the components of each link profile together with the neighborhood of each node, in order to better evaluate the cause of packet losses. The basic idea is for the nodes whose IDS reported packet drops, to re-profile their links with the suspicious node and compare them to the CHPs for those links (generated during initial setup and periodically updated during the lifetime of the sensor application). The link re-profiling algorithm is carried simultaneously by several investigating nodes and without the knowledge of the suspicious node, in order to avoid any misdirection with the investigation results.

[2]In this dissertation, we refer to the term interference for both intentional (jamming) and unintentional disruption of signal communication between sensor nodes.

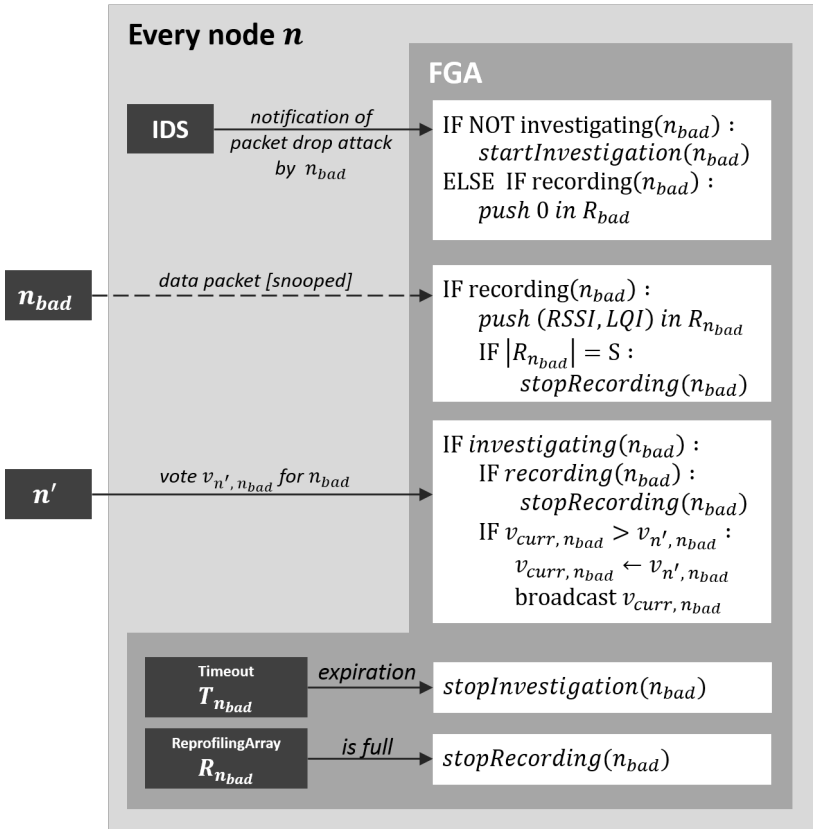


Figure 6.2. FGA event-driven algorithm.

Our FGA algorithm is event-driven, and thus ideal for asynchronous systems such as WSNs. The triggering events and their respective actions are shown in Figure 6.2. Basically, when the IDS of a node, say n, observes packet drop attacks at one of its neighbor nodes, say n_{bad}, it will activate the FGA tool of n in order to investigate the cause of the packet drops observed at n_{bad}.

In order to assess the current health status of the link with n_{bad}, the investigating nodes must collect new data evidence. The FGA tool of n therefore starts the *Evidence Collection* phase, snooping packets sent by n_{bad} and recording their corresponding RSSI and LQI values in a re-profiling array $R_{n_{bad}}[]$ of size S. For every subsequent notification by the IDS of n that another packet has been dropped by n_{bad}, a value of 0 is pushed into $R_{n_{bad}}[]$. This process is carried out in parallel at every node that observes packet drops from n_{bad} and is investigating accordingly.

While the FGA tool of a node is investigating, it also simultaneously carries out an *environmental evidence collection* in order to better account for the surrounding environment. In fact, the FGA tool records the RSSI and LQI measurements for every snooped packet coming from other direct neighbors. Similarly to what is performed for n_{bad}, values coming from a neighbor, say n', are pushed into the corresponding re-profiling array $R_{\mathrm{n'}}[]$, up to S elements.

6.4.2 Profile Comparison

At the time, say TC, when one of the investigating nodes fills up its re-profiling array $R_{n_{bad}}[]$, this node, say n, computes a *Current Investigation Profile (CIP)*, representing the most current health state of the link between itself and n_{bad}, as follows. First, all re-profiling arrays (for n_{bad} as well as the other direct neighbors of n) are aggregated similarly to what is done during the initial profiling, thus generating a single profile for each link with a neighbor n' from which some data was snooped during the Evidence Collection phase:

$$
\begin{aligned}
CIAvgRSSI_{n'} &= \frac{\sum_{i=1}^{S} RSSI\left(R_{\mathrm{n'}}[i]\right)}{|S|} \\
CIAvgLQI_{n'} &= \frac{\sum_{i=1}^{S} LQI\left(R_{\mathrm{n'}}[i]\right)}{|S|} \\
CIPRR_{n'} &= \frac{nonzero\left(R_{\mathrm{n'}}[]\right)}{S} \\
CIP_{n'} &= \langle CIAvgRSSI_{n'}, CIAvgLQI_{n'}, CIPRR_{n'} \rangle
\end{aligned}
$$

Note that, since every packet loss originated by n_{bad} detected during the Evidence Collection pushes a 0 in the re-profiling array $R_{n_{bad}}[]$, the Packet Reception Rate for the link $n \leftarrow n'$ in this investigation on n_{bad} is easily computed as the ratio between the number of non-zero values in the array and the size S of the array.

All the current profiles $CIP_{n'}$ (including $CIP_{n_{bad}}$) are compared to the respective $CHP_{n'}$ to compute a series of deltas $\triangle_{n'}$ indicating the absolute change in profile for each of these neighbor nodes, for the single components of RSSI and LQI:

$$\triangle_{RSSI_{n'}} = abs(RSSI(CIP_{n'}) - RSSI(CHP_{n'}))$$
$$\triangle_{LQI_{n'}} = abs(LQI(CIP_{n'}) - LQI(CHP_{n'}))$$
$$\triangle_{n'} = \langle \triangle_{RSSI_{n'}}, \triangle_{LQI_{n'}} \rangle$$

Once all the deltas are calculated, all the $\triangle_{n'}$ except $\triangle_{n_{bad}}$ are averaged into a Neighborhood Current Investigating Delta $NCID$. Finally, the Current Investigation Delta CID is computed as follows:

$$CID = \alpha \cdot \triangle_{n_{bad}} + \beta \cdot NCID$$

where $\alpha + \beta = 1$ and $\alpha \geq \beta$.

The parameters α and β are customizable depending on how much weight the environment should have in the final overall profile for that link. Including the environment in the Profile Comparison makes it harder for a malicious node to fake an interference by intentionally manipulating the transmission power, as discussed later in the security analysis (see Section 6.8). However, it is easy to see that the effect of the rest of the environment on the Investigation Profile can be removed by choosing $\alpha = 1.0$ and $\beta = 0.0$. Experiment 7 in Section 6.7 provides insights on appropriate values for the two weights.

At this point, the node executes the Profile Comparison algorithm, a fundamental building block in our analysis technique. Figure 6.3 shows a representation of the decision flow for this algorithm. The algorithm compares $CIPRR_{n_{bad}}$ against the threshold (PRR_{thres}) that differentiates between the cases of partial and total packet losses. We denote the single RSSI and LQI components of the CID as $\triangle_{RSSI}$ and $\triangle_{LQI}$, respectively. $Interf_{thres}$ denotes the interference threshold representing the

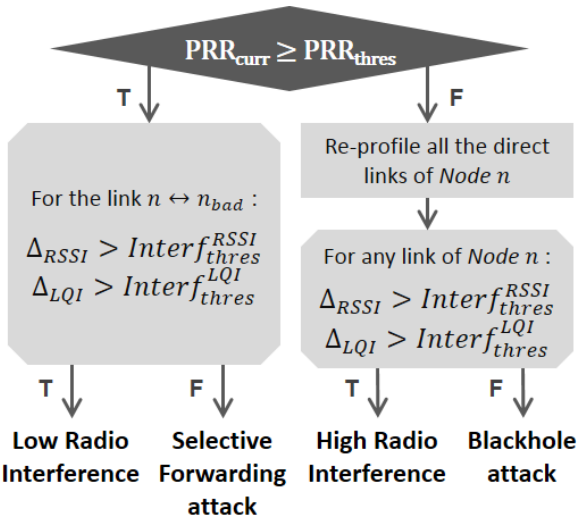


Figure 6.3. Profile comparison algorithm at *Node n* being an neighbor node investigating for packet drops observed at node n_{bad}.

minimal difference in link profiles (RSSI, LQI) that can determine the existence of interference. The value of $Interf_{thres}$ is determined by measuring the maximum fluctuation occurred during the initial profiling of the network links.

When $PRR_{curr} \geq PRR_{thres}$, the profile comparison algorithm of each neighbor node would only need to compare the link profiles that connect it with the node under investigation. However, when $PRR_{curr} < PRR_{thres}$, the profile comparison algorithm of each neighbor node would need to re-profile all its links and compare the new profiles with the originally captured link profiles.

6.4.3 Threshold Values Determination

The FGA Profile Comparison algorithm leverages two separate steps of thresholding. How to determine the best thresholds is very different for the two levels.

The PRR_{thres} value differentiates between what should be considered a total packet loss from what is only a partial packet loss. As such, it is very application-dependent. In fact, consider two different WSN applications, as one that collects temperature samples from a crop and another that monitors the health status of a

hospital patient, both of which send out packets every 1 minute. In the first application, only having, for example, 30% of the packets delivered correctly to the base station could be slightly inaccurate but sufficient to plot the temperature trend in that crop; the temperature will not change too quickly, and the application is not overall mission-critical. On the other hand, in the second application, even losing just $10 - 20\%$ of the packets (thus obtaining $80 - 90\%$ of the packets delivered correctly at the base station) can be extremely dangerous: the health status of the patient can change very drastically in a short span of time, and the criticality of the WSN application leads to defining even a small packet loss as total loss. For these reasons, the two application might want to set their PRR_{thres} values to 0.3 and 0.85, respectively, and both choices would be perfectly legitimate and reasonable in light of their requirements. Anyway, as part of our future work, we plan on investigating techniques to support users in choosing a PRR threshold. A possibility involves verifying the potential presence of redundancy mechanisms. For example, for mission-critical applications, data packets could be duplicated and simultaneously sent along multiple disjoint routes to the BS; in this case, the PRR local to a link is a much less pressing constraint given the redundancy. Moreover, measuring the overall WSN packet delivery success rate under normal conditions could provide a better understanding of a tolerable packet loss rate that can still maintain network functionality.

Unlike the PRR_{thres} threshold, the values of the $Interf_{thres}$ threshold are not application-dependent, and are calculated from the measures collected during the initial profiling. In fact, the rationale behind an optimal value for the $Interf_{thres}$ is to minimize the misdetections (link-related attack diagnosed as node-related, and viceversa) during the investigation. The RSSI and LQI values are subject to natural fluctuations, the strength of which is determined by the motes used and various environmental factors. During the initial profiling, many sample RSSI and LQI values are collected. The optimal threshold to minimize misdetections is determined by measuring the maximum fluctuation occurred during the initial profiling of the network links, separately for the two dimensions of RSSI and LQI, excluding strong

outliers (i.e. values outside the $95th$ percentile). That maximum fluctuation effectively establishes how much the RSSI and LQI can deviate from the initial profile value without the presence of any attacker-induced interference, and correctly delivering the packet (i.e. not causing a packet loss). To consider a concrete example, consider a node that collects some initial profiling packets for a specific link, and calculates the initial profile for that link as the average of the values collected, say $RSSI = 78$ and $LQI = 105$. However, assume that the RSSI and LQI values of the collected packets range in $[76, 82]$ and $[104, 106]$, respectively. Then, it is easy to see that legitimate values for those two dimensions can fluctuate up to 6 points for the RSSI and 2 points for the LQI, without being the result of a malicious interference. A value outside of these bounds, during a FGA, will indicate a divergence greater than the natural one experienced during the initial profiling, and thus indicate an interference attack. Therefore, the values for $Interf_{thres}$ can be calculated directly based on the maximum fluctuation experienced by a node during the initial profiling.

6.4.4 Majority Voting and Investigation Results

When the first node to finish the evidence collection completes the Profile Comparison, it is ready to vote on what the most likely cause of packet drops at n_{bad} is. The resulting vote is broadcasted with a message with the following format:

$$\langle n_{bad}, NodeIDs[], Vote, TC \rangle$$

where $NodeIDs[]$ is an array that initially contains the ID of the first voting node but eventually will contain the IDs of all voting nodes.

Every other node that is still investigating on n_{bad}, say n', will stop recording packets for this investigation on n_{bad} at the time of receiving the first broadcasted vote. Moreover, it will only consider those collected packets with timestamp less than or equal to TC. This is necessary because up until the first vote is broadcasted, the investigation was hidden to n_{bad}, and only after TC the suspect node will become

aware of the undergoing FGA investigation. The suspect node may therefore attempt to alter its sent packets to misdirect the investigation.

The majority voting algorithm is designed as a lightweight, simple Distributed Agreement protocol. Every node initially computes its own vote according to the evidence collected and the Profile Comparison algorithm already discussed (see Figure 6.3). Only nodes with enough packets recorded in $R_{n_{bad}}[]$ according to a preset threshold will take part in the voting. Every node that computes its vote (its own or aggregated) will broadcast the vote and start a timeout $T_{n_{bad}}$. Any node that receives a vote with the $NodeIDs[]$ array including new node ID(s) will consider the received vote as more updated and therefore will aggregate this with its own vote (if not already included) and then broadcast such new aggregated vote. $T_{n_{bad}}$ is reset every time a vote is broadcasted. All votes received from n_{bad} are ignored and do not reset $T_{n_{bad}}$. When $T_{n_{bad}}$ expires at a node, the node considers the vote aggregation complete, meaning that no more votes are circulating. In addition, every neighbor node will have the most updated final vote result, with $NodeIDs[]$ listing the IDs of all the nodes that broadcasted this vote. Therefore, the appropriate action might be taken depending on the policy put in place by the administrator. For example, an automated Intrusion Response System [43] might take action, or the BS might be notified of the determined attack.

Vote Aggregation. We first present the intuitions behind the vote aggregation mechanism, and then we introduce a formal definition. At a high level, the vote aggregation is based on the following rules:

- **High Radio Interference**: If at least one node votes for *High Radio Interference*, the aggregated vote is high radio interference and therefore packet drops at n_{bad} are link related. A node would vote for *High Radio Interference* when the RSSI and LQI values of its neighborhood profile are significantly affected due to such strong interference.

- **Low Radio Interference**: If at least one node votes for *Low Radio Interference* and none of the other nodes votes for *High Radio Interference*, the aggregated vote is low radio interference and therefore packet drops at n_{bad} are link related. A node would vote for *Low Radio Interference* when the RSSI and LQI values of the link profile with n_{bad} are significantly affected, without significant changes in the overall neighborhood profile.

- **Selective Forwarding Attack**: If at least one node votes for *Selective Forwarding* and none of the other nodes votes for any type of interference in the network medium, the aggregated vote is selective forwarding and therefore packet drops at n_{bad} are node related. A node would vote for *Selective Forwarding* when none of its received packets from n_{bad} has any significant changes in their RSSI and LQI values when compared with its original link profile connecting both nodes.

- **Blackhole Attack**: If at least one node votes for *Blackhole* when none of other investigating nodes votes for selective forwarding or any type of interference in the network medium, the aggregated vote is blackhole and therefore packet drops at n_{bad} are node related. A node would vote for a blackhole attack when none of its neighborhood link profiles are affected by interference nor any packets have being forwarded from n_{bad}.

Interference might affect different, close-by links with different strengths. For this reason, initial votes by different nodes might be diverse. However, the node aggregation effectively places a total ordering on the votes, always considering a vote for a link-related attack more reliable than a vote for a node-related attack. In fact, if some nodes detect interference while some others do not, it is most likely that interference is present, but not affecting the communications of some of the investigating nodes. Either way, interference will still be the cause for the packet droppings and should therefore be chosen as the correct final decision.

We now formalize such intuition for the vote aggregation mechanism. Let the domain of votes be:

$$D = \{HI\_IN, LO\_IN, SEL\_FWD, B\_HOLE\}$$

such that each element represents *High Radio Interference*, *Low Radio Interference*, *Selective Forwarding* and *Blackhole*, respectively. We define a total order relation over D as follows:

$$HI\_IN \prec LO\_IN \prec SEL\_FWD \prec B\_HOLE$$

Given a set V of votes to be aggregated, the result is computed through the aggregation function defined as follows:

$$aggr(V) = min(V)$$

Since the RSSI and LQI values may be subject to sampling bias that could alter the voting result of a single node, we achieve better voting accuracy in network topologies that allow each node to have at least two direct neighbors and more than one disjoint path to reach the BS. This redundancy is typical in engineering a real-world sensor network and therefore makes our voting results less diverse.

Communication Complexity and Optimizations. In the majority voting algorithm just described, every time a node receives a new vote, it aggregates it to its current local vote and immediately rebroadcasts it. The final, exact number of messages sent by the investigating nodes depends on the specific topology of the neighborhood where the majority voting algorithm is taking place, but it is easy to see that, in the worst case, the algorithm has a communication complexity of $O(n^2)$ in terms of messages exchanged among n investigating nodes. An optimization that we developed as a performance enhancement of the FGA tool is able of reducing the

communication requirements of the majority voting algorithm. For this optimization, the execution of the majority voting is locally divided into rounds (a notion common in many distributed agreement protocols). The round will be determined exclusively by the local timeout $T_{n_{bad}}$, that will have a different use in this version of the algorithm. When a new vote is received, it is aggregated with the current local vote, but it will not be re-broadcasted right away. Instead, the investigating node will keep collecting and aggregating all the votes it receives during the current round. Every time the timeout fires, the execution will advance to the next round. If any new votes were received in the previous round, the node will broadcast the latest aggregated vote in its possession. If, instead, no new votes were received during the previous round, the majority voting phase is considered completed and the node uses the final vote that was computed, as in the old version of the majority voting. Even for this optimized algorithm, the exact number of messages sent by the investigating nodes depends on the specific topology of the neighborhood where the majority voting algorithm is taking place. However, this time we can imagine that a new vote travels at least one additional link per round, and at most one message is sent per node per round. Therefore the algorithm has a communication complexity of $O(l)$, where l is the longest hop distance between two nodes participating to the investigation. Since the worst case topology for this communication is a linear structure with all the nodes linked in a chain, the longest hop distance for n nodes is $n - 1$ hops; therefore the communication complexity worst case would be $O(n)$.

6.5 Colluding Investigating Nodes

The majority voting algorithms presented so far are based on the adversarial model presented in Section 6.1, that assumes the investigating nodes to be trustworthy and not to report false votes. Under such assumptions, a single vote can change the final outcome of the collective investigation. In fact, as already discussed and motivated, the vote aggregation algorithm places a total ordering on the values in the domain

of possible attacks, and chooses the minimum value as the new aggregated vote. Therefore, for example, a single vote for a link-related attack counts more than any number of votes for a node-related attack, according to our adopted ordering.

We now relax our adversarial model to allow for colluding nodes among the investigating nodes. Under this model, the vote aggregation algorithm needs to be revised, since allowing a single vote to potentially change the result might let a malicious investigating node to easily subvert the final decision. We therefore present an algorithm effective in scenarios where some investigating nodes might be colluding with a malicious node and thus enacting Byzantine behaviors. For every investigation, the algorithm is able to tolerate up to f colluding investigating nodes, with the total number of investigating nodes is at least $f + 1$.

For this algorithm, we use the notion of *multiset* as a data structure for a collection of votes. Intuitively, a multiset is a set in which each element is associated with its multiplicity. More formally, a finite multiset Θ over our vote domain D is a function

$$\Theta : D \to \mathbb{N}$$

that is nonzero for finitely many $v \in D$. The cardinality of a multiset Θ is calculated as $\sum_{v \in D} \Theta(v)$. The minimum of a non empty multiset Θ is computed as

$$min(\Theta) = min\,\{d \in D \; s.t. \Theta(d) > 0\}$$

Through this definition of minimum, we can introduce the function $l(\Theta)$ that defines the multiset obtained by removing one occurrence of the smallest value in Θ as follows:

$$l(\Theta)(v) = \begin{cases} \Theta(v) - 1 & \text{if } v = min(\Theta) \\ \Theta(v) & \text{otherwise} \end{cases}$$

With these definitions in place, we can proceed to describe the algorithm steps. In this enhanced algorithm, every vote message is broadcasted with the following format:

$$\langle n_{bad}, Votes[] \rangle$$

where $Votes[]$ is an array of individual votes, each specified as:

$$\langle NodeID, Vote, TC \rangle$$

This means that every vote message can contain multiple votes proposed by different investigating nodes, whose ID is specified in the $NodeID$ field. When a node first finishes the evidence collection phase and the subsequent Profile Comparison, its vote will be the only one included in the vote message it broadcasts. Then, the algorithm operates in rounds, like the optimized algorithm described in the previous section. Every time a node receives a vote message, it merges the contained votes with the ones it has already collected. Every node collects all the votes it receives during a particular round r, and then broadcasts a vote message containing all the votes it has collected so far in the following round $r + 1$.

The vote collection terminates when, in a particular round, no new votes are received. At this point, all the votes collected so far are placed in the multiset. In order to tolerate f colluding nodes that might try to skew the investigation result, we intuitively need to remove some of the votes that could alter the final outcome. For this, we reduce the multiset by applying the function $l(\Theta)$ iteratively f times, thus removing the f lowest instances of votes in the multiset. At this point, the final decision for a node given the votes that it has collected during the majority voting phase and adjusted to tolerate colluding nodes is computed as:

$$decision = min(\Theta)$$

This decision can be then, for example, communicated to the BS or reported to the IRS, according to the policy put in place by the network administrators.

6.6 Locating Interference Sources

In critical (real-time) sensor applications, it is necessary to be able to anticipate possible future attacks on the network. This is achieved by enriching the system with learning techniques that can warn of possible attacks, based on previous faulty scenarios. In cases when radio interference is causing packet losses, it is necessary to locate the source of the interference in order to identify the network region (set of nodes/links) that may be also affected by the noise source.

Locating the source of interference is necessary as it may assist in evaluating the trust level of sensor readings. When interference is the cause of packet losses at a certain node or link, its effect may also reach other regions where the IDS might require further diagnosis. This could leave the whole network in an inconsistent state for a longer period of time, spent between detection and analysis of the same cause. Therefore it is necessary for the FGA tool to locate the source of interference and inform the BS about the affected region as part of its analysis. Even though the FGA tool is decentralized and does not rely on the BS for detecting the cause of packet drops, it will however require the BS's knowledge of the network map in locating the source of interference.

6.6.1 Design Choices for Localization

Localization algorithms can be classified as centralized and distributed [178]. In centralized approaches, all the localization data for all nodes are collected and processed centrally – usually at the BS – with a global overview of the entire network. On the other hand, distributed approaches use information from the node itself and the direct neighbors. Therefore, centralized approaches lead to a more accurate localization, but require more communications. Thus, whereas for this reason the distributed approach is usually preferred for WSNs, our FGA approach benefits from a hybrid approach that carries out part of the computation at the investigating nodes (such as the collection of re-profiling samples and the computation of the deltas for localiza-

tions), while delegating to the BS the main computation. This strategy is particularly appropriate for our work since the localization will need to be communicated to the network administrator anyway in order for her to intervene.

In our work, as well as in the state-of-the-art approaches, the accuracy requirements for WSN localization approaches depend on many factors, including: the requirements of the application itself, the dynamics of the monitored environment, cost and energy consumption, the availability of additional supporting hardware, the required speed of the computational processing. The requirements that drive the design decisions in our approach include: no need for any additional hardware, maximization of the reuse of already-collected data, minimal processing for faster completion of the localization, low energy consumption impact, enough accuracy to pinpoint the area affected by interference and guarantee administrator intervention. To better understand the rationale behind our design choices, it is important to formalize that the goal of our interference localization is to give direction to in-network countermeasures, either automated – by Intrusion Response Systems that can for example re-route packets around the affected area – or manual – by a network administrator that wants to physically find and remove the malicious source of interference. These two goals, driven by the needs of real-world scenarios, drive the tradeoff between the speed and simplicity of the algorithm, and an exact position localization. In realistic scenarios, this justifies a potentially small inaccuracy – inherent in the signal strength measurement in WSNs – in order to quickly respond to the attack and to preserve energy.

In addition, our approach has three main advantages compared with traditional signal strength-based localization approaches:

- In traditional RSSI-based localization techniques, a node A collects on-demand some samples of signal strength received from a node B, which as we discuss in this chapter may fluctuate quite a bit. In our approach, the link profiles that each node records are the result of an accurate samples collection, are smoothed

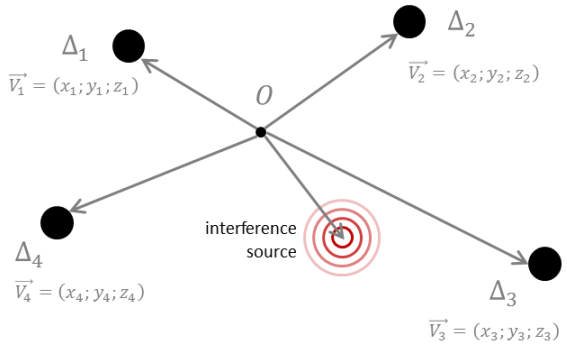


Figure 6.4. V_i and $\triangle_i$ representing *node* i's location and neighborhood profile delta.

out to account for the natural signal fluctuations, and are constantly kept up to date during the network's lifetime when energy is discharging.

- In such traditional techniques, node A then uses its own collected samples to estimate the distance of B. In our approach, the information used for the localization process comes from multiple investigating nodes, providing our approach with redundancy and thus more robust and accurate. Moreover, the aggregation of such data for the localization process is carried out at the BS, that having more resources can afford to store additional information such as the complete, detailed map of the WSN.

- The most crucial part of our technique is that, instead of trying to estimate the distance from a node, the investigating nodes only report to the BS the relative deltas in all their link profiles, that is, how affected by the interference these links are. So, instead of measuring how strong the signal is, this is a measure of how attenuated it is with respect to a normality profile built over time. The BS, then, based on the map of the WSN, can use those deltas as weights to each link in its search, and accurately localize source of interference.

6.6.2 Localization Approach

When the FGA tool determines that the cause of packet losses is an interference, it attempts to locate the interference source in order to estimate the nodes and links that may be possibly affected. The approach used for detecting the interference source works as follows. Let n_{bad} be the first node reported to the BS as affected by interference. The BS then requires all the direct neighbors $N_{n_{bad}}$ of this node to re-compute their current neighborhood profiles, as discussed in Section 6.3. Once the new profiles are computed and compared to the original neighborhood profiles, each node sends the BS the profile difference, denoted as $\triangle$, for evaluation.

The technique used to locate the source of interference leverages the formula for the calculation of the weighted centroid of finite points [179], defined as:

$$\frac{\sum_{i=1}^{n} W_i \cdot \vec{V_i}}{\sum_{i=1}^{n} W_i}$$

where W_i is the weight at *Node i* computed as a function of $\triangle_i$, and $\vec{V_i}$ is the vector with the spatial coordinates $(\vec{x}, \vec{y}, \vec{z})$ of *Node i* according to the actual topology of the network known by the BS. Figure 6.4 also shows the vectors that represent each node's location according to a given origin O and their corresponding $\triangle$. The BS will use these $\triangle_i$s and $\vec{V_i}$s, together with knowledge of the topology, to locate the interference source.

6.6.3 Weight Function

As we already mentioned, the weight W_i used in the weighted centroid of finite points formula for the localization of interference sources is computed as a function of $\triangle_i$. Depending on the particular function chosen, the localization accuracy could vary greatly. In order to determine the weight function that would guarantee the best localization accuracy, we carried out an extensive analysis, applying different weight functions to all the experimental data collected from our real-world testbed.

Figure 6.5 shows a comparison of the percentage error of some of the different weight functions we used. As the results show, we determined that the function that guarantees the best accuracy in all our tests is the *exponentiation* function. Therefore, this is the function of choice for our FGA localization algorithm, and is the function used in the localization experiments described in Section 6.7.

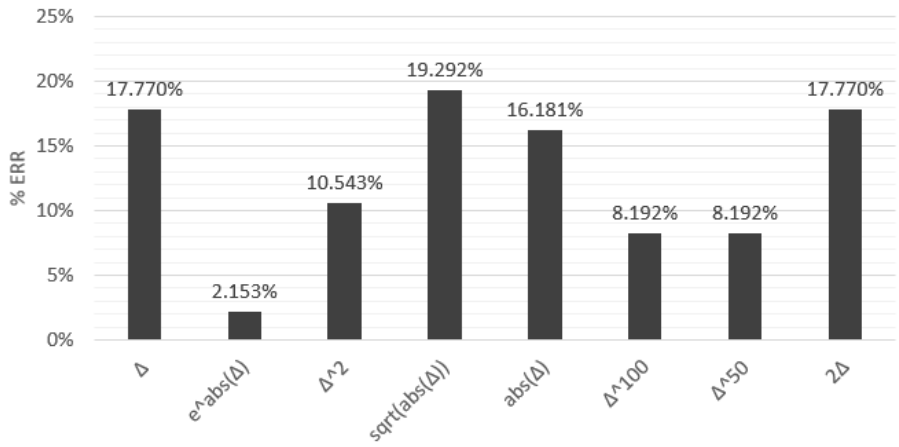


Figure 6.5. Comparison of accuracy with different functions for the weight in the weighted centroid of finite points formula for the localization of interference sources (smaller is better).

6.7 Experimental Analysis

In this section, we report experimental results to assess the efficiency and accuracy of our FGA tool. We first introduce our experimental setup, then we perform real-world experiments to test our FGA tool on different attack scenarios.

6.7.1 Experimental Setup

Our setup consisted of 25 TelosB [180] wireless sensor motes using *TinyOS 2.1*, which were placed at different locations. These motes operate at 2.4 GHz ISM band, with an effective data rate of 256 kbps, a much higher rate than that of older radios.

For experimental purposes, we set up one sensor to act as a BS and created a server Java program to interact with the BS through the USB port. All the other nodes were programmed with the same code and waited for commands from the BS. Each node was programmed to perform its analysis locally and independently.

We also built a simple routing system on top of the standard messaging layer that offers point-to-point multi-hop direct communication. We applied concepts from common Internet routing protocols so that every node is capable of automatically building its own routing table and self-discovering/learning the best routes towards any other node in the network. Thus, after a few initial packet exchanges, most of the routes are automatically discovered and so is the whole network. Our improved routing system reduces the nodes communication overhead by avoiding useless packet transmissions. Through the BS, we used the multi-hop protocol to allow the administrator to send commands to all the nodes, as well as to single specific nodes, through our Java program that was connected to the BS.

We also set up a special mote, the jammer, to act as the source of interference in order to test the FGA accuracy in detecting and locating the noise [181]. This mote was programmed to emit dummy packets every 5 *msec.* with an increasing counter value. We avoided keeping the other mote radios busy with useless interrupts, while maximizing the interference on the radio medium itself. This helped us to get a more realistic representation of a real-world interference attack.

The accuracy of the FGA is related to the considered topology of the sensor network. The minimum topology requirements would be for each node to have at least two possible paths to reach the BS. This requires that every link has at least one additional node watching it, forming a triangular structure for every hop towards the BS. This requirement is basic for most real-world sensor networks, considering that redundancy is always required by the engineering of every topology.

The testbed network topology for our experimental evaluation follows this structure, in which the links form many triangular shapes.

Figure 6.6 shows a snapshot of the topology of a portion of our network that is the closest to the BS. Focusing on a portion of the network is effective in showing the performance of the FGA tool, since investigations are always local to the neighborhood of a detected packet loss. We performed experiments using this topology to compute the corresponding profiles of each link. Then we used these profiles to detect the causes of packet drops as detailed in Section 6.4.

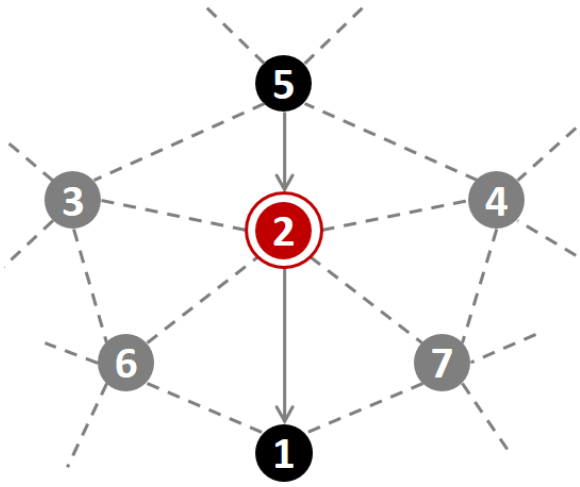


Figure 6.6. Snapshot of the network portion closest to the BS.

6.7.2 FGA Testing on Different Attack Scenarios

We carried out our testing at different parts of the network to better evaluate its efficiency, and we experimented all possible cases by placing the interference source at different locations.

The results we present in this section come from 5 independent repetition of each experiment in identical conditions.

Since only the direct neighbor nodes participate in the FGA of a suspect node, we will refer to Figure 6.6 in our experiments for clarity of discussion, as attacks causing packet losses will occur among these nodes.

		To Node																							
		2				3				4				5				6				7			
		RSSI		LQI		RSSI		LQI		RSSI		LQI		RSSI		LQI		RSSI		LQI		RSSI		LQI	
		I	C	I	C	I	C	I	C	I	C	I	C	I	C	I	C	I	C	I	C	I	C	I	C
From Node	2					-55	-72	106	105	-62	-64	107	107	-58	-62	107	106	-62	-70	107	104	-55	-60	108	107
	3	-58	-86	107	106					-58	-65	107	106	-64	-83	107	107	-59	-72	106	105	-55	-60	108	107
	4	-68	-67	107	107	-55	-61	107	107					-67	-68	107	106	-59	-59	107	106	-58	-60	107	107
	5	-60	-64	108	107	-62	-76	107	107	-64	-66	107	105					-59	-62	106	105	-57	-58	107	107
	6	-67	-75	107	107	-60	-67	106	106	-57	-59	107	106	-63	-63	107	107					-59	-59	107	107
	7	-62	-65	107	106	-54	-59	107	107	-60	-62	107	106	-57	-58	107	105	-58	-58	107	105				

Figure 6.7. Comparison of link profiles for nodes 2, 3, 4, 5, 6 and 7 with and without interference near node 3. "I" denotes initial profiles values, and "C" denotes current profile values in the presence of interference.

Our experiments were divided into several scenarios:

Experiment 1: Building Profiles at Initial Network Setup: At initial network setup, the BS requests every node to start building its initial profiles with its direct neighbors, and save these profiles locally at every node. The time needed to profile our 25 sensor nodes, which each sends 100 *dummy* messages to each other node every 10 milliseconds, was a total of 25 sec., thus 1 sec. to profile each node links. Figure 6.7 shows sample profile values we collected from *Nodes 2, 3, 4, 5, 6 and 7* (see Figure 6.6).

Experiment 2: Interference Effect on Link Profiles: In order to test the impact of interference on link profiles, we activated our interference mote and placed it close to *Node 3*. We manually requested re-profiling to see the changes in the RSSI and LQI values of the affected links, mainly around *Node 3*. Figure 6.7 shows the corresponding profiles of these links. Notice the changes in the RSSI values with no significant changes in the LQI values due to low radio interference we purposely used. Also, since the LQI values that $CC2420$ radios report are independent from the RSSI parameter, the link quality is stable once the signal strength is good enough.

Experiment 3: Selective Forwarding Attack: To test the FGA tool against this attack, we configured our network so that *Node 5* would send messages to the BS, whereas *Node 2* would be the intermediate node as chosen by the routing protocol. *Nodes 3, 4, 6 and 7* would be the direct neighbors of *Node 2* and thus monitor its behavior.

The intermediate *Node 2* was programmed to simulate a selective forwarding attack by dropping packets with 10% probability, and we made sure that there was no interference within range. Our results show that when 20 packets were dropped by *Node 2*, the IDSes of *Nodes 3, 4, 6 and 7* successfully detected a packet drop attack and triggered the FGA. In our experiment, *Node 4* was the first to completely fill its re-profiling array of 10 slots at time TC, compute, and broadcast its vote. Each of the *Nodes 3, 6 and 7* had already recorded the RSSI and LQI values of 9 received packets from *Node 2* at the time stamped in *Node 4*'s vote. The FGA tool of each node in turn aggregates and broadcast its vote accordingly until all votes of the investigating nodes are aggregated. The final vote aggregation reported a selective forwarding attack.

Vote for node 3: $\langle 2, [3], SEL\_FWD, TC \rangle$

Vote for node 4: $\langle 2, [4], SEL\_FWD, TC \rangle$

Vote for node 6: $\langle 2, [6], SEL\_FWD, TC \rangle$

Vote for node 7: $\langle 2, [7], SEL\_FWD, TC \rangle$

Aggregated vote: $\langle 2, [3, 4, 6, 7], SEL\_FWD, TC \rangle$

Experiment 4: Low Interference Attack: We placed the interference mote to carry out low interference near *Node 2*. The interference however was not strong enough to isolate *Node 2* completely. However, when 20 packets were dropped by *Node 2*, the IDSes of *Nodes 3, 4, 5, 6 and 7* detected a packet drop attack and activated their FGA tool. *Node 3* was the first to fill its re-profiling array slots and send out its vote. *Nodes 4 and 6* received the vote and stopped recording with 9/10 slots filled, while *Node 5 and 7* stopped recording with 8/10 slots filled. Since our

threshold for accepting a node's vote is to have at least 70% of its re-profiling array slots filled, we considered the votes of all 5 investigating nodes. Even though *Nodes 4 and 7* voted for "Selective Forwarding" as their recorded packets did not show any significant influence of the existing interference, *Nodes 3, 5, and 6* voted for "Low Interference" which is the aggregated vote result according to the methodology we presented in Section 6.4.

Vote for node 3: $\langle 2, [3], LO\_IN, TC \rangle$

Vote for node 4: $\langle 2, [4], SEL\_FWD, TC \rangle$

Vote for node 5: $\langle 2, [5], LO\_IN, TC \rangle$

Vote for node 6: $\langle 2, [6], LO\_IN, TC \rangle$

Vote for node 7: $\langle 2, [7], SEL\_FWD, TC \rangle$

Aggregated vote: $\langle 2, [3, 4, 5, 6, 7], LO\_IN, TC \rangle$

Experiment 5: Strong Interference Attack: This attack scenario was the same as the previous attack scenario but we strengthened the interference to such a point that *Node 2* was completely isolated from the network. *Nodes 3, 4, 6 and 7* carried out the analysis with *Node 4* filling out its re-profiling array first and broadcasting its vote. *Nodes 6 and 7* received the vote and stopped recording with 7/10 slots filled, while *Node 3* with only 5/10 slots filled. Since our threshold for accepting a node's vote is to have at least 70% of its re-profiling array slots filled, we did not consider the vote of *Node 3*. Because of the applied strong interference, the re-profiling arrays of *Nodes 4, 6 and 7* contained 0's and therefore had to re-profile their links with their direct neighbors to perform the profile comparison algorithm. Even though *Node 6* was the only node to report "High Radio Interference", its vote was dominant according to the methodology in Section 6.4.

Vote for node 4: $\langle 2, [4], B\_HOLE, TC \rangle$

Vote for node 6: $\langle 2, [6], HI\_IN, TC \rangle$

Vote for node 7: $\langle 2, [7], B\_HOLE, TC \rangle$

Aggregated vote: $\langle 2, [4, 6, 7], HI\_IN, TC \rangle$

Experiment 6: Locating the Source of Interference: In this experiment, we applied the technique described in Section 6.6 to locate the source of interference that was causing packet losses. We placed the interference source at different locations, mainly near *Nodes 2, 3 and 4*, to compute the accuracy of our location method. Figure 6.8 shows the actual and estimated location of the interference source when placed near *Node 3*. Once the BS receives the neighborhood profile difference, it can locate the interference source using its precise/approximate coordinates of each node. Figure 6.9 shows the accuracy of our method by comparing the actual location of the interference source to the computed ones resulted from various interference locations. We present three of the most relevant test cases for this experiment. In each of the tests, the source of interference was placed in a different position (indicated as "Actual Position"), and we computed the error with respect to the "Estimated Position" as the Euclidean distance of the two points. In Test 1 and Test 2, we placed the source of interference quite close to two different nodes, respectively, while in Test 3 we placed it in the middle of a link. The particular weight function we use for interference location, as discussed, is the exponentiation of the neighborhood profile delta. Such function, while guaranteeing the lowest error among all the functions we used, tends to "pull" the estimated position next to the nodes; this is the reason why Test 3, while still proving very accurate, showed a higher error than the other two tests.

Experiment 7: Power Manipulation: A smart malicious node might try to misdirect the outcome of an investigation by manipulating its transmission power after dropping a packet. By using the investigation profile for the malicious node alone, the investigating nodes might detect a variation in the profiles greater than the threshold for interference, and thus determine a link related attack instead of a node related attack. We carried out an experiment to verify that the introduction of the Environmental Evidence Collection, as well as its use in calculating the Current Investigation Delta, can prevent such misdirection. We used the same topology as

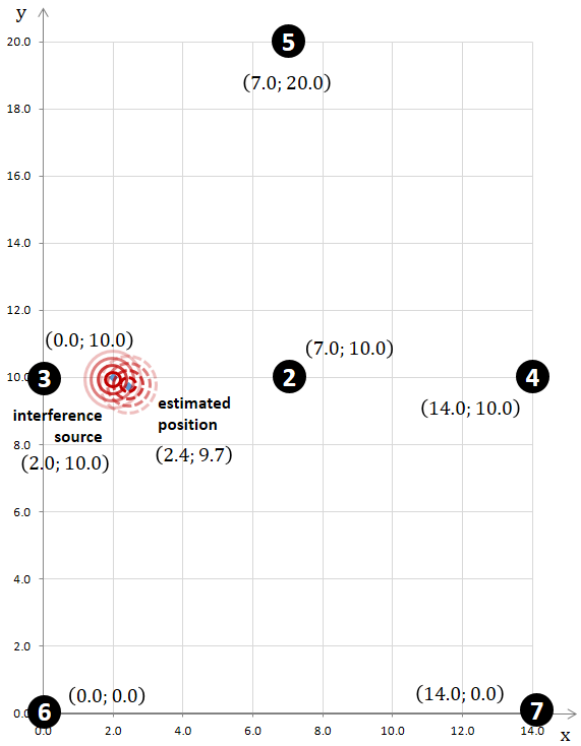


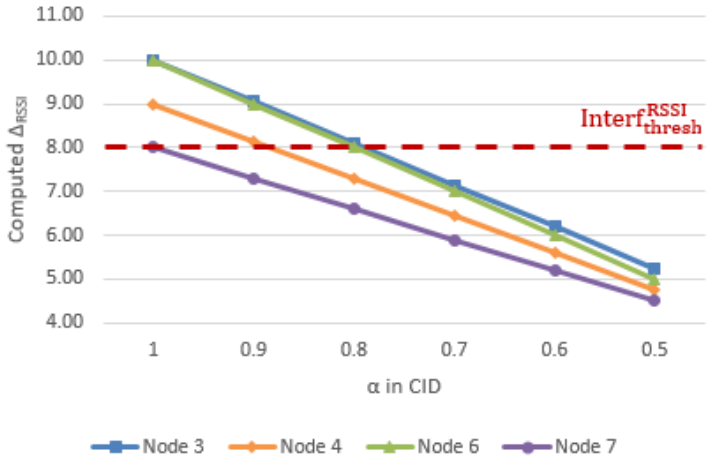
Figure 6.8. Actual *vs.* computed location of an interference source with respect to sensor nodes, in feet.

Test 1	Actual Position	(14.0; 10.0)
	Estimated Position	(13.8; 10.1)
	Error	**0.773%**
Test 2	Actual Position	(14.0; 20.0)
	Estimated Position	(14.0; 19.9)
	Error	**0.288%**
Test 3	Actual Position	(2.0; 10.0)
	Estimated Position	(2.4; 9.7)
	Error	**2.153%**

Figure 6.9. Accuracy of computed *vs.* actual location (in feet) of different interference source positions.

the other experiments, instructing *Node 2* to carry out a selective forwarding attack and, immediately after, lowering its transmission power by $10dBm$ during the inves-

tigation by the neighboring nodes. We detected that this power manipulation caused, on average, a variation of 9.25 in the RSSI component of the profile of n_{bad} ($\Delta_{n_{bad}}$) at the investigating nodes. Figure 6.10 shows how the computed Current Investigation Delta varies at different investigating nodes depending on the values chosen for α and, consequently, β. For ease of analysis, we present the results for the RSSI components of every delta, as well as of the $Interf_{thres}$, which for this experiment was set to 8.0 based on the fluctuation measured at network setup. The highlighted cells show the occurrences of detection of interference instead of selective forwarding, indicating that the misdirection by the attacker succeeded. The results show that a value of 0.8 for α already changes the outcome for 2 out of 3 nodes that would have decided for an interference attack using the profile for n_{bad} alone. With a value of α set to 0.7 or lower, all the nodes correctly identify the attack despite the power manipulation.



		Values for α in CID					
		1	0.9	0.8	0.7	0.6	0.5
Investigating node	Node 3	10.00	9.05	8.10	7.15	6.20	5.25
	Node 4	9.00	8.15	7.30	6.45	5.60	4.75
	Node 6	10.00	9.00	8.00	7.00	6.00	5.00
	Node 7	8.00	7.30	6.60	5.90	5.20	4.50

Figure 6.10. Accuracy of the detection in presence of power manipulations using different values for α.

Experiment 8: Tolerance to Colluding Investigating Nodes: In this experiment, we tested the tolerance of our collusion-resistant majority voting algorithm in presence of malicious investigating nodes, presented in Section 6.5.

We repeated the scenario described in Experiment 3, programming one of the intermediate nodes to perform a selective forwarding attack. We also programmed one of its neighbor to act as a colluding node during all the investigations, and thus to broadcast fake votes.

At first, we did not use the collusion-resistance algorithm. Once the investigation started, the nodes started broadcasting correctly their votes for "Selective Forwarding". The colluding node, then, broadcasted its fake vote for "Low Interference". Since that vote, according to our total ordering, is less than the other votes circulating, the other nodes chose it as a voting result and incorrectly reported a low interference attack to the BS as the cause of the packet dropping. Therefore, the collusion attack was successful.

We then repeated the same scenario but using our collusion-resistant majority algorithm. This time, after all the votes were collected, they were placed into the multiset and the "Low Interference" vote, being the lowest, was discarded. The investigation result was therefore correctly computed as a selective forwarding attack. Thus, the collusion attack in this case was prevented, tolerating for the malicious investigating node.

Experiment 9: Initial Profiling Stability for Sample count: In this experiment, we evaluate how the duration of the initial profiling in terms of exchanged dummy packets affects the obtained profile stability. We have carried out the initial profiling procedure, as already discussed in this work, by exchanging up to 1000 dummy packets (in each direction) between each pair of node in our testbed.

We characterize a profile by its mean, maximum and minimum values, which will describe the value for the profile and the fluctuation. Moreover, we define a profile

as *stable* when any number of future samples does not change the mean, maximum and minimum by more than 1.0.

Figure 6.11 shows the statistics for one of those links, focusing on the RSSI. On the x axis there is the number of samples collected, while the y axis shows the signal strength value for each packet. It is easy to see that after a sample size of 60, all the three statistics do not change by more than 1.0, therefore we consider the profile as stable.

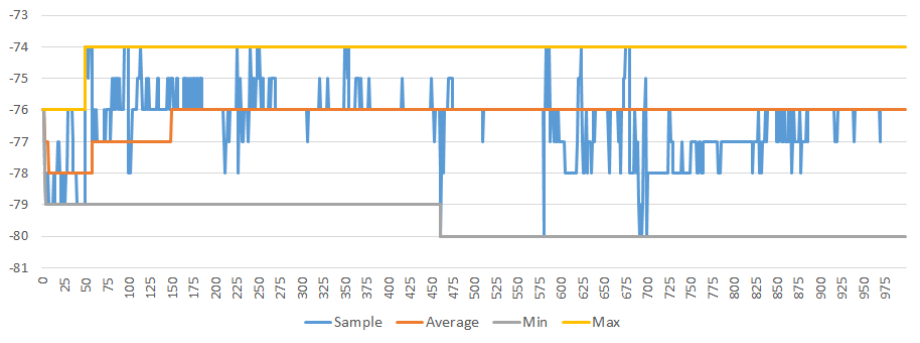


Figure 6.11. Relationship between initial profiling duration and profile accuracy.

We carried out this same evaluation on all the links of our testbed, to determine the necessary number of sample until the respective profile may be considered stable.

Overall, our results show that the minimum number of dummy packets necessary, during the initial profiling, to obtain a stable link profile ranges between a minimum of 26 and a maximum of 74, averaging at 43, and longer profiling times do not add significant advantages.

Summary of Results: Figure 6.12 shows the accuracy of our FGA technique with respect to the actual number of packet loss test cases we performed. Our FGA technique was able to perform correct diagnoses in $\sim 90\%$ of the cases when selective forwarding was the cause of packet losses (*Experiment 3*), in $\sim 95\%$ of the cases when low interference was the cause (*Experiment 4*), and in $\sim 100\%$ of the time when strong

interference was the cause of packet losses (*Experiment 5*). These results show the accuracy of our FGA technique for sensor network applications.

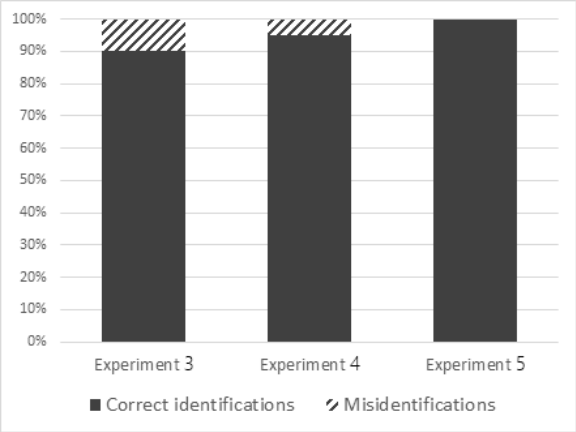


Figure 6.12. Comparison of FGA accuracy in experiments 3, 4, and 5.

6.8 Security Analysis

In this section, we analyze the security of the FGA technique with respect to the possible strategies that an adversary might use to hide the cause of packet drops. The aim of our security analysis is to devise mitigations against such strategies and discuss how likely they are to compromise the results of the FGA.

Transmission Power Manipulation. When a node is suspected to be maliciously dropping packets, the FGA tool executes the re-profiling algorithm as described in Section 6.4. However, an attacker may try to misdirect the FGA by using a different power than the usual power to transmit the messages included in the re-profiling process, which will make the neighbor nodes think the packet loss is caused by link interference. However, our re-profiling procedure is hidden from the attacker and thus the node under attack will not be able to differentiate between normal and re-profiling traffic. Only after the FGA results (votes) are broadcasted, the attacker might become aware that the re-profiling process has already been executed.

A smart attacking node might still, however, manipulate its sending power for the packets following a node-related packet dropping attack. For this reason, the investigating nodes also collect evidence about the environment (i.e. the rest of the traffic coming from their neighbors) during the investigation, and this data is used in the profile comparison to reduce the possibility of success of an attack with intentional transmission power manipulation.

Organic Battery Decay vs. Interference. We now consider the scenario in which an attacker tries to trick the FGA tool by very gradually and slowly introducing interference until the network becomes disrupted. The Profile Update algorithm would detect this change as an organic battery decaying and not interference, thus leaving the attack undetected. However, if the network administrator sets the threshold rate for the change smoothness of accepted decaying (i.e. the threshold for the rate at which the change is organically gradual) to a low, appropriate value, then the attacker would only be able to produce as much interference as what the natural decaying of the overall network battery life is. Therefore, the attack would be effectively harmless. The same considerations arise in the scenario in which a single node would alter its power levels slowly enough to be considered physiological battery discharge. Again, the attacker would only be able to decrease the level as slowly as the natural decaying of the node's battery, thus producing a very slow and effectively harmless attack.

Strong Interference and Disruption of the FGA. When the WSN is under a strong interference attack, if the attack covers an area larger than a single/few nodes, it might disrupt communication for a whole node neighborhood. Therefore, even assuming that some of the nodes in the neighborhood could start investigating, the interference might prevent some of the FGA tool's fundamentals steps, such as the exchange of votes. If no interference-free node exists, and the entire network is blocked out, then in this worst case the FGA tool would not be able to complete the

analysis but no recovery would be possible anyway. In any well-designed WSN, the BS profiles the traffic over time and, in a case like this, it would notice a huge difference in the traffic profile and immediately detect the attack. However, there will often be a frontier around which the effect of the jammer is not strong enough. If, thus, such a frontier exists (as it will in most of the cases, not having interference strong enough to disrupt an entire WSN), the fully distributed FGA algorithm guarantees that the nodes around that frontier will be able to investigate successfully.

Correctness of Initial Link Profiles. One possible interference source could be a result of closely placed sensors that could disturb the network communication signals. After a successful network initial setup, the existence of strong interference among the sensors is easily detectable. However, very low interference could exist that may affect the RSSI values but without causing any packet drops. The initial link profiles are collected right after a successful network initial setup. Thus the RSSI values collected may be affected by the existing interference. However, we claim that those link profiles are still valid and correct as long as no packets drops have been detected by the IDS during the initial collection process. Moreover, the existence of a malicious node among the newly installed nodes is unlikely as we believe that each node would be tested at initial setup, thus also validating the correctness of the FGA initial profiles.

Reactive Jamming Attacks. In a reactive jamming attack, the jammer may modify the Start-of-Frame Delimiter (SFD bytes) of some packets before sending them, resulting in dropping packets at the destination node. As such packet modification does not increase $\triangle_{RSSI}$ or $\triangle_{LQI}$, the FGA algorithm might think that the destination node is malicious, but actually it is not. As packet modification is out of the scope of this dissertation, we rely on the IDS to check if there is any packet modification before the FGA tool starts carrying its re-profiling algorithm.

Simultaneous Different Attacks at Same Location. Our FGA approach is able to carry out multiple simultaneous investigations about attacks in different parts of the network. However, there could be extreme instances in which a fully correct diagnosis would not be possible right away. In fact, in the case of multiple attacks of different nature happening at the same time at the same node – i.e. when a node-related and a link-related attacks occur jointly in the same location – the interference attack would always affect the surrounding links in a way that can mask attacks from compromised nodes. Out of the two attacks occurring simultaneously at the same node, therefore, the FGA would be able to initially diagnose only the interference attack, as the changes in the link profiles would hide the further attacks by the compromised node. However, this would still lead the network administrator to be able to diagnose and resolve the interference attack in that portion of the WSN. Then, if the attacks from the compromised node are still happening, these would be detected correctly by the FGA upon the immediately following packet loss event, and would lead the administrator to completely eradicate the problem.

6.9 Related Work

Our work is related to previous research on two different topics: (1) the use of forensic analysis techniques for investigating packet losses in WSNs, (2) the use of RSSI/LQI for WSN performance and sensor localization.

Very few forensic analysis techniques have been proposed to investigate packet losses occurring in networks. Yang *et al.* developed a detection scheme employing neighboring nodes as witnesses that monitor their peers for possible misbehaving nodes incorrectly forwarding packets [40]. The forensic analyzer by Ning *et al.* aims at determining the cause of discarded packets and forwarding misbehaviors by means of various network parameters (packet size, bit rate in use, node density, interference level) and logs [41]. Our approach, in contrast, uses less network parameters and fewer computations to determine the likely causes of packet losses. Moreover, our

approach differentiates between node and link related causes of packet drops, while their approach focuses on differentiating between natural induced packet losses from malicious discarding.

Several approaches have been proposed for detecting packet dropping attacks, but few approaches identify the cause of packet drops through their impact on network parameters, as we investigate with our FGA technique. Ramach *et al.* proposed a generic architecture able to monitor many parameters in protocols, devices and networks parameters [37]. On the same line, the diagnostic system introduced by Qiu *et al.* leverages trace-driven simulations in order to diagnose performance problems caused by various network adverse events, such as link congestion, packet dropping, MAC misbehavior and external noise [36]. However, the evidence collected by both these approaches aims at diagnosing performance issues, rather than determining the most likely cause of packet losses. De Couto *et al.* designed an expected transmission count metric to estimate the packet delivery ratio on links [38], similar to the ETT metric that assigns weights to individual links based on the expected transmission time of a packet over the link [39]. Although both the metrics assess the packet loss rate, neither of them gives information about the cause of packet losses.

The RSSI and LQI metrics of the $CC2420$ have also been used for goals different from ours. Zaruba *et al.* used RSSI readings for locating wireless nodes in an indoor environment, requiring a single access point [182]. However, Parameswaran *et al.* [183] determined that the sole RSSI metric is sufficient for localization algorithms, but they were not always successful in getting very accurate measurements of node distances due to the presence of factors such as interference. Zanca *et al.* [184] compared many RSSI-based localization algorithms and showed that by just using RSSI, localization may not be accurate with errors of few centimeters due to the presence of moving people or obstacles. However, Srinivasan and Levis [185] argued that the combination of RSSI and LQI measurements represents an effective indicator for localization, even with the existence of obstacles or interference. Other uses of RSSI parameters were proposed by Khan *et al.* for troubleshooting unresponsive sensor nodes. However,

their approach requires the use of external power-metering subsystems located next to the nodes to collect their power consumption traces in case of future possible failures [172].

In [186], Bocca *et al.* use RF sensor networks for real-time device-free localization, by leveraging the change in RSSI of the various links to infer the targets locations. Their technique is similar to the one in our proposed approach, even though more geared towards real-time tracking of moving targets, and their 0.5 m average error demonstrates how signal strength-based localization approaches can achieve a high accuracy.

Chen *et al.* in [187] propose a signal strength-based localization algorithm for WSNs by using *dependable* RSSI values, chosen with empirically defined thresholds. While our approach is similar, and leads to comparable accuracy, we add the flexibility of achieving the same level of dependability in RSSI values by smoothing down the samples collected during link profiling. Therefore, in our approach there is no need to pre-determine a dependability threshold for the samples.

In [188], Bekcibasi *et al.* discuss the classification of factors that reduce RSSI accuracy for localization purposes as environmental and device factors. The former are related to the wireless communication channel itself, and include multipath, shadowing and interference. The latter are related to the devices radio chip and include, for example, calibration errors. In our work, the discussed way of our approach for building and updating the link profiles is able to account for both these categories of inaccuracy factors: the initial profiling strengthens against environmental factors, while the use of relative deltas for localization instead of absolute values protects against device factors.

Some traditional localization approaches make use of *anchors*, nodes that will measure the received signal strength from the target (often moving) and use it to triangulate the position by intersecting the radiuses of multiple anchors. Bekcibasi *et al.* propose in [188] to use groups of four anchors instead of three in order to improve the localization accuracy. In our approach, all the investigating nodes effectively act

as anchors – adding even more redundancy to the measurements – but without the burden of knowing their own exact location – which is delegated to the BS.

We originally proposed the FGA technique in [189]. In the current status of the work, we enhance our technique by addressing some limitations of the original approach. A first limitation was that, over the execution life of a WSN, the use of a static reference profile for the links is not able to account for the organic battery decay. In our current work we address such limitation by the introduction of the Current Health Profile to reduces the possibility of false positives. Sophisticated adversaries might intentionally manipulate the transmission power in order to mask node-related attacks as interference issues. By performing the Environmental Evidence Collection, the improved FGA technique includes reference data that accounts for the surrounding links in order to prevent those power manipulations from misdirecting the investigation. In this work, the enhanced security provided by the improved FGA tool is investigated in more depth, to analyze different adversarial scenarios such as colluding nodes or strong interference preventing local investigations.

6.10 Summary

This chapter introduced our FGA technique for wireless sensor networks that uses existing link parameters to investigate the cause of packet losses, whether it is related to node attacks or to link interference attacks. Our technique has been implemented in a tool that has been deployed on actual sensors. Experiments on these sensor nodes have shown that our FGA technique is able to successfully differentiate the various attacks and determine the most likely cause of packet losses. Also, in the case when interference is the cause of packet losses, our FGA technique is able to locate the interference source and to estimate its effect on other nodes and links.

Since determining the correct parameters to the diagnostic system can be a hard task, in the next chapter we present a statistical model to enhance the FGA system and assist in determining the necessary thresholds.

7 STATISTICALLY-ENHANCED FINE-GRAINED DIAGNOSIS OF PACKET LOSSES

All the parameters for the FGA tool, such as its detection thresholds, can be customized by the network administrator based on the requirements of their specific WSN of interest. Incorrectly setting the parameters can impact the accuracy of the analysis and, consequently, the correctness of the packet loss cause determination in face of subtle attacks. While the parameter values identified in Chapter 6 were evaluated to be effective, an automated guidance to the choice of optimal values for such system parameters is of great importance. In fact, empirically-determined values might not always be optimal. As a consequence, a higher number of false alarms would be produced, reducing the accuracy of the tool. Moreover, such previous approach uses a single threshold for the whole WSN. While this is aimed at reducing the workload for the network administrator, it can lead to an additional increase in false alarms. In fact, in a large-scale WSN deployment, different parts of the network might experience different normality conditions, and a single predefined threshold might be suitable for a network portion but inadequate for another. Lastly, the previous approach does not allow one to control the false alarm rate for each link. This means that it is not possible to require a priori a desired maximum rate of false detections. We therefore design an approach that builds and uses a statistical model for the determination of the optimal system thresholds. By collecting and analyzing samples from the initial deployment of the WSN system, our approach builds an accurate statistical model of each link exploiting the variances of RSSI and LQI. Based on such model, our approach is able to select the optimal threshold for each link in the WSN. One of the advantages of our model is that it also allows the setting of a different threshold for each link. Such task is manually unfeasible for a network administrator, but can be effectively carried out in an automated way by our model.

Moreover, since each threshold is tuned according to an optimum criterion, we can always choose a desired false alarm rate on a per-link basis and, if needed, exclude from the network all the links that will not be able to reach satisfactory detections. By means of extensive MATLAB simulation based on real sensor data, our experimental evaluation shows that our model is accurate and leads to well-tailored system parameters for an optimally-accurate fine-grained analysis of the underlying causes of packet losses.

7.1 System Model

7.1.1 WSN Metrics Formalization

In this section, we provide a formal definition of well-known resident packet metrics and aggregated value that we leverage for building our model. We refer to the TelosB motes [57] as our hardware platform of choice. Such platform, specifically, uses a *CC2420* radio chip, but our approach can be applied to any of the newer radios based on the IEEE 802.15.4 standard. In fact, these radio chips natively offer two measurements for link quality estimation, namely RSSI and LQI. The RSSI represents an estimate of the received signal power for a packet, and it is measured in dBm. Its value is calculated over an 8-symbol period, long on average 128 s. The dynamic range for the RSSI is between -50 and -100, with higher values (less negative) representing a stronger signal. It is worth noting that, on the CC2420 chip, the manufacturer specifies that the read RSSI value is stored in the `RSSI_VAL` register of the chip, with a fixed offset of -45 dBm. According to the specifications of the IEEE 802.11.4 standard, the RSSI value can be effectively used for both detecting noise on a channel, and estimating the quality of an incoming packet upon its reception. This property is leveraged by many protocols for optimal routing decisions [124], and validated by several research efforts on the accuracy of the RSSI measurements themselves [185, 190]. The LQI can be seen as the chip error rate of the received signal and measures the signal reception quality. It is calculated over the 8 bits after

the start frame delimiter (SFD). The specifications of the CC2420 radio chip state that the measured LQI is actually the average correlation of each symbol obtained by comparing the symbol that is supposed to be received and the symbol actually received (signal plus noise). LQI values range between 110 and 50, corresponding respectively to maximum and minimum quality frames. Another important metric for link quality estimation is the PRR. It is not a value natively computed by the radio chip, but instead an aggregated metric for each individual link, computed as the ratio between the number of packets successfully received and the number of packets sent. Higher values of PRR indicate a better link quality and therefore a healthier communication medium.

$$PRR = \frac{\#successfullyreceivedpackets}{\#sentpackets}$$

7.2 A New Profiling Technique

In this section, we first discuss the motivations behind the new profiling approach, and then show the rationale of our procedure.

7.2.1 Motivations

The profiling proposed by in Chapter 6 has proven to be very effective in discriminating between a wireless network affected by a packet drop attack (either selective forwarding or blackhole attack) or by low/high interference (i.e. noise). One of the major drawbacks of this method is that the threshold is only one for the entire network and, above all, the threshold is empirically evaluated (i.e. not tuned according to an optimum criterion). Here, we move further by proposing a new profiling technique that, by exploiting the variances of the RSSI and LQI parameters, can define a threshold for each link in the network. The problem is formulated as a conventional binary hypothesis test, where the two hypotheses H_0 and H_1 correspond, respectively, to the absence or presence of the attack of interest. We define as probability of false

alarm (P_{FA}) the probability of our technique to declare the presence of the attack when it is actually not present. Conversely, we define as probability of detection (P_D) the probability of our system to correctly identify the presence of the attack, when it is actually performed. To limit the computational cost of the decision device, we choose some one-dimensional testing variables that are compared to a pre-selected threshold to efficiently perform the test. The optimal threshold for each link is tuned according to an optimality criterion [191]. In particular, the constant false alarm rate (CFAR) procedure, typically used to perform effective tests [192], is here exploited as such an optimality criterion. The CFAR procedure refers to a common form of adaptive algorithm used in telecommunications systems to discriminate between the presence and absence of something (e.g. an unknown user in hidden communications), against a background of noise and interference [193]. The CFAR criterion is executed, for each link, according to the following two steps: first, the threshold is determined that limits the false alarm probability at a given reduced value (i.e. the size of the test) under the null hypothesis H_0; then, the probability of detection, P_D, (i.e. the power of the test) is evaluated under the alternate hypothesis H_1 for the threshold previously determined. The main idea behind our procedure is that the variance of the received signal (and hence the variance of the RSSI and LQI of the received packets) is lower when the link is affected by a packet drop attack (H_0 hypothesis), while it is higher in the alternate H_1 hypothesis (i.e. when the link is affected by low/high interference). In fact, and as noted previously, packet drop attacks do not change the statistics of the received signal in terms of RSSI and LQI. Conversely, the same consideration is not any longer valid in the presence of channel noise. Therefore, we can select the variances of the received RSSI and LQI as the one-dimensional testing variables to compare with the optimal thresholds. Hence, our variance-based test can be effective in discriminating between cases of a packet drop attacks by nodes and the presence of low/high interference. In addition, since these considerations apply to each link in the network, we can identify the most appropriate threshold and consequently the detection performances for each network link.

7.2.2 Rationale

The rationale behind our method is the following. The Intrusion Detection System (IDS) has declared the presence of an anomaly (i.e. some kind of attack) in the network. This declaration is based on the observation of the PRR values. This allows us to discriminate between the following two hypotheses:

- H_0: the WSN is affected by a packet drop attack;
- H_1: the WSN is affected by an interference attack.

The decision about the presence or absence of an interference attack in a certain link of the WSN can be obtained by comparing the decision metric to the pre-selected threshold. In the case of our interest, the decision metric (or decision variable) is the variance of the RSSI and LQI of the received packets on that link. Let $u(n)$, $i(n)$ and $x(n)$ be the sequences (of N samples) representing, respectively, the (useful) transmitted signal, the interference affecting the communication in the network, and the received signal. Since node-related packet dropping attacks do not change the signal statistics, the problem can be formulated as follows:

$$H_0 : x(n) = u(n); H_1 : x(n) = u(n) + i(n) \tag{7.1}$$

Then, assuming that the signal and the interference are zero-mean, mutually independent random processes, the two hypotheses result in:

$$H_0 : {}^2_x = {}^2_u; H_1 : {}^2_x = {}^2_u + {}^2_i \tag{7.2}$$

where ${}^2_u$ and ${}^2_i$ are the variances of the useful signal and the interference respectively, while ${}^2_x$ is the variance of the received signal. The variance ${}^2_x$ can be the variance of either the received RSSI values or of the LQI values. For the sake of the compactness, in what follows we refer only to the test in terms of RSSI variance. The same considerations apply to the LQI case. Now, the estimation of the variance of the

received RSSI values is used as the testing variable for discriminating about the presence of a packet drop or interference attacks. The new testing variable is estimated according to the following expression:

$$\hat{Z} = \frac{1}{N}\sum_{n=1}^{N} x(n) - E[x]^2 \tag{7.3}$$

Then, considering a threshold η, the test is finally formulated as follows:
Packet drop attack: $Z < hat\eta$; Interference attack: $Z >= hat\eta$

This means that, if the testing variable is greater than the threshold value ($\hat{\eta}$), then the algorithm decides for the hypothesis H_1 (i.e. interference attack), otherwise the choice is for the hypothesis H_0 (i.e. packet drop attack). Finally, the testing variable in 7.2.2 is asymptotically ($N \rightarrow$ inf) Gaussian as a direct consequence of the central limit theorem. Hence, the test threshold can be asymptotically tuned from a straightforward evaluation of the Gaussian integral for a fixed probability of false alarm, under the null-hypothesis [193]:

$$\eta = E[\hat{Z}] + \left(\sqrt{2 \cdot var[\hat{Z}]}\right) \cdot erf^{-1}(1 - 2P_{FA}) \tag{7.4}$$

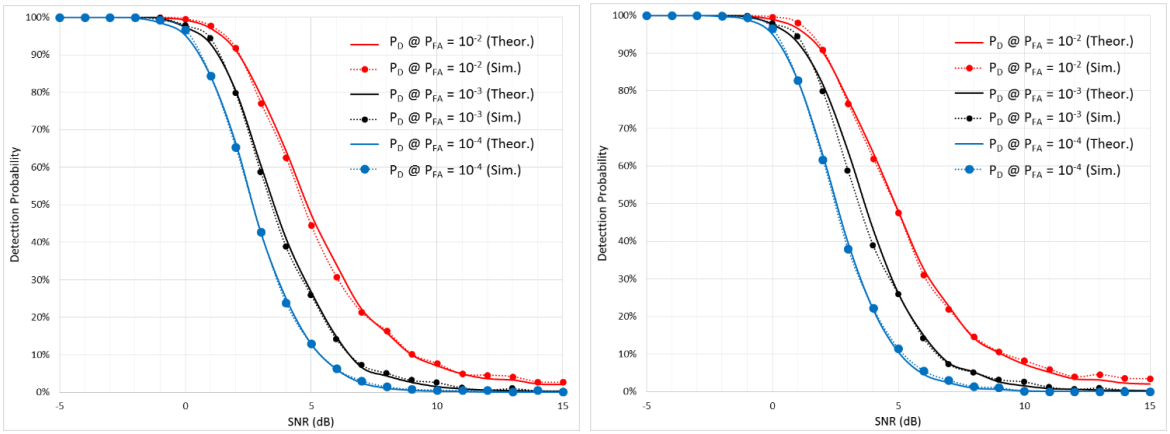
where $E[\hat{Z}]$ and $var[\hat{Z}]$ denote the expectation and variance of the testing variable, respectively, while $erf^{-1}(\cdot)$ is the well-known (inverse of the) complementary error function. It has to be noted that the time-consuming threshold setting stage is usually performed off-line (i.e. during the deployment of the network, when no attacks are occurring). Then, the thresholds are pre-computed and stored on look-up tables for several SNR values. Finally, the probability of detection P_D is determined in the H_1 hypothesis as:

$$P_D = 1/2 + 1/2 \cdot erf\left(-\eta + \frac{E[\hat{Z}]}{\sqrt{2 \cdot var[\hat{Z}]}}\right) \tag{7.5}$$

7.3 Evaluation Results

In this section, we report experimental results validating our theoretical approach and assessing the efficiency of the proposed profiling technique. The WSN setup consists of 16 TelosB sensors placed in a 4x4 grid. These nodes use the CC2420 radio chip, natively providing the RSSI and LQI measurements for each received packet. In our experiments, we have collected a large amount of real sensors data under two operating scenarios of interest. In the first scenario, the WSN works properly without any interference (H_0 hypothesis); in the second scenario, a sensor acting as a jammer introduces different levels of interference in the WSN communications. The interference varied from low, to medium and high values in order to collect data at several signal-to-noise ratios (SNR) of practical interest. We have first estimated the variance of the received RSSI and LQI parameters according to 7.2.2, and then tuned the optimal theoretical threshold using 7.2.2.

It is important to mention that the number of testing variables, used to estimate the mean and variance in 7.2.2, directly impacts on the threshold tuning. More testing

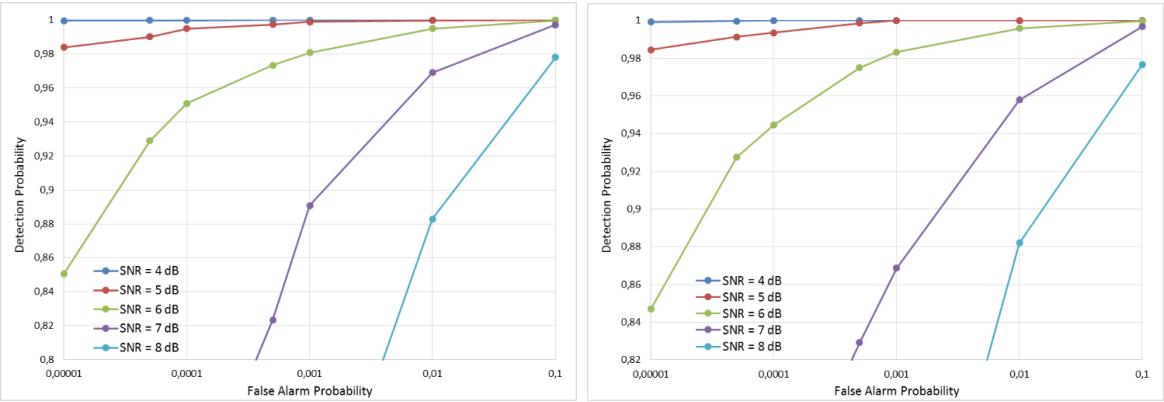


(a) RSSI variance (b) LQI variance

Figure 7.1. Theoretical (Theor.) and experimental (Sim.) probability of detection of the proposed method for several values of SNR and different false alarm probabilities exploiting: a) the RSSI variance; b) the LQI variance. Simulation (dotted lines); theory (solid lines).

variables allow one to set a finer threshold corresponding to the P_{FA} targets (according to the CFAR procedure), while fewer testing variables result in a rougher threshold setting with an actual P_{FA} not according to the CFAR procedure. We thus decided to work with 1000 testing variables, estimating the variance of the received RSSI/LQI values over $N = 1000$ samples, since this value represents a reasonable trade-off between test performance and accuracy. Then, we have evaluated the performance of our test for three different false alarm probabilities (i.e. $P_{FA} = 10^{-2}$, $P_{FA} = 10^{-3}$, and $P_{FA} = 10^{-4}$). We have evaluated P_D both analytically, i.e. using 7.2.2, and experimentally. Figure 7.1 shows the performance of our profiling technique in terms of the detection probability of an interference attack. In particular, Figure 7.1(a) refers to the observations of the RSSI variance, while Figure 7.1(b) illustrates the case of the LQI variance. The simulation results (dotted lines) well match the theoretical ones (solid lines), thus validating the correctness of the mathematical analysis and assumptions. As the SNR increases (from -5 dB to 15 dB, i.e. the attacker changes the interference level from high to medium-low interference), the performance of the proposed FGA technique decreases, as expected. In fact, if the level of the interference by the attacker (or jammer) is too low, it becomes impossible for the sensor to discriminate between interference and a packet drop attack. However, notice that we are able to obtain a true detection (higher than 90%) in the presence of a low interference (at about SNR $= 3 - 4$ dB), even at very low false alarm probability (i.e. $P_FA = 10^{-4}$).

Finally, in order to fully assess the performance of our profiling method, we have investigated how the detection and false alarm probabilities relate to each other. The receiver operating characteristic (ROC) curve illustrates the performance of our binary profiling method as its discrimination threshold is varied. The ROC curve is created by plotting the detection probability against the false alarm probability at various threshold settings. Ideally, all the ROC curves must be above the line $P_D = P_{FA}$ (bisector) and concave downward. Paradoxically, if they were not, a randomized test would be better. The best performing detector presents the minimum



(a) RSSI variance (b) LQI variance

Figure 7.2. Theoretical ROC curves for several values of SNR exploiting: a) the RSSI variance; b) the LQI variance.

distance from the ideal point ($P_D = 100\%$ and $P_{FA} = 0\%$) in its ROC curve. An effective operating point is just the point of the curve near such an optimum case. Figure 7.2 shows the theoretical ROC curves, i.e. P_D vs. P_{FA}, of our profiling technique for several values of the SNR. In particular, Figure 7.2(a) refers to using the RSSI variance for the test, while Figure 7.2(b) refers to the use of the LQI variance-based test. For the sake of the simplicity, only the theoretical curves are reported in Figure 7.2, since the experimental data again perfectly overlap the theoretical ones. It is interesting to note that our method is able to identify a true detection (with a $P_D > 90\%$) even in the presence of an interference attack with a SNR of 7 dB, allowing the target P_{FA} to increase from 10^{-4} to 10^{-2}. In other words, our test needs to work with lower SNR values, to maintain the same level of detection, thus decreasing the false alarm rate. As we can see from the previous graphs, larger detection probabilities are achieved also in the presence of (low-power) interference attacks, thus demonstrating the effectiveness of our test for fine-grained diagnosis of packet losses in wireless sensor networks.

7.4 Summary

This chapter presented an approach that builds a statistical model for optimally-accurate fine-grained analysis of the underlying causes of packet losses in WSNs, whether node- or link-related. Our model exploits the variances of RSSI and LQI to determine an individual, optimal detection threshold for each link. Unlike other techniques such as Artificial Neural Networks (ANN) and Nave Bayes, our approach is extremely fast and requires less computational resources, allowing implementing it on the sensors. In addition, the proposed approach allows us to have control on the P_{FA} through the CFAR criterion. Extensive MATLAB simulations, based on real sensor data, validated the accuracy of our model and the optimality of its system parameters under the constant false alarm rate criterion.

With all the accurate diagnostic information on the detected security incidents, it is possible to enact an effective recovery. However, automating such response is a challenging task, and we address this issue in the work presented in the next chapter.

8 A SYSTEM FOR RESPONSE AND PREVENTION OF SECURITY INCIDENTS IN WIRELESS SENSOR NETWORKS

It is vital that WSNs services be continuously provided even in face of anomalies or attacks, and to effectively recover from attacks without significant interruption. Detection alone is not enough, but the monitoring and diagnostic techniques presented in the previous chapters provide crucial information useful to quickly react to the attacks, by taking actions that make the system able to continue its operations and at the same time to block the attacks. We thus need response tools that would enable automatic responses and recovery actions. The intrusion response systems developed for other domains, such as database systems or distributed systems, cannot be directly used in WSNs due to significant differences in their operations, resources, and communication. An effective intrusion response system tailored to the characteristics of WSNs would need to fulfill the following requirements:

- **Lightweight:** The constrained resources of WSN nodes call for a system that is lightweight in terms of both computational cost and resource usage. In particular, the response policies should be specified in a flexible but simple fashion, so not to incur much overhead when selecting the appropriate response actions.

- **Real-time:** While the response system must be effective in ensuring that the WSN operations are not interrupted by adverse events, it should at the same time execute the most effective action for each anomaly or attack in a secure fashion.

- **Cooperative:** To fulfill the two previous requirements, as well as make sure that the response system itself is resilient to attacks, the intrusion response system should have a fully distributed design. It should rely on local opera-

tions and cooperative strategies, instead of heavy interactions with a central authority.

In this Chapter, we present *Kinesis* – the first systematic approach to a security Incident Response and Prevention System (IRPS) for WSNs. The system is lightweight, cooperative, and distributed. According to our design, each sensor in the WSN is a watchdog monitor [29] and hosts both an IDS, and the Kinesis system. Kinesis best works in static WSNs, which are a very common scenario for WSN applications (e.g., open-air deployments for agriculture domains, indoor deployments for healthcare scenarios, ...). The Kinesis operations are based on a three-phase workflow:

- **Detect.** Through the IDS, the monitor observes neighbor behaviors, detects suspicious incidents (anomaly/attack) in the neighborhood, and notifies Kinesis.
- **Diagnose.** While IDSes are generally capable of detecting anomalies and attacks, they might not be aware of the root causes for such adverse events. Therefore, in some instances further diagnosis is needed. This task is executed by Kinesis in order to have a better understanding of the events in the network.
- **Respond.** Upon being notified of an incident, and after acquiring sufficient knowledge about the occurred incident and the current security state of the neighbors, Kinesis matches the appropriate response policy from the set of response policies specified by the base station (BS). The system thus performs the most appropriate response actions to pursue its security goal.

To support a flexible specification of response policies in Kinesis, we propose a WSN-specific lightweight policy language based on the Event-Condition-Action (ECA) paradigm [194]. In a response policy, a set of rules maps incidents and anomalies to different response actions to be performed, based on security assessments of the suspect node. A monitor estimates the security level of the suspect node based on the (i) incident detection confidence, (ii) suspect's behavior history, and (iii) incident impact on the WSN. This strategy helps in selecting the most effective response

action. We have surveyed the various attacks in WSNs and created a taxonomy of attacks (Figure 8.1) and a comprehensive set of response actions (Table 8.3). However, Kinesis can generate responses against an unknown attack based on the anomalous behavior the attack manifests.

To trigger the response execution corresponding to an incident, Kinesis selects a *daemon* node in a neighborhood via a self-organized competition among the neighbors. The competition is controlled in a distributed fashion by a per-node *action timer*. The node whose timer fires first wins the competition and executes the action. Most of the actions involve a transmission which is overheard by the neighbors and then allows the neighbors to stop their action timers and to refrain from taking redundant actions. Thus, Kinesis does not require any message exchanges for the response action synchronization and has no communication overhead. A node's action timer value is locally estimated based on: (i) neighborhood size, (ii) neighbor link qualities, (iii) time since its last action. It reflects the effectiveness of a node in executing the action and ensures load distribution among the neighbors.

The distributed nature of Kinesis also enhances security. When a node is compromised, other legitimate nodes in the neighborhood can continue with the Kinesis functionalities. Kinesis is secure in terms of policy dissemination and storage since the BS specifies the policies, converts them to a binary code and disseminates the binary throughout the network with a secure dissemination protocol [195].

Following, we summarize our **contributions:**

(1) We build Kinesis, the first (to the best of our knowledge) IRPS for WSNs able to continue the WSN services despite an attack or anomaly and to recover from the attack eventually. We present the design and architecture of Kinesis, as well as a prototype implementation in TinyOS.

(2) Considering the context and constraints of WSN, we propose a lightweight policy language to express the response policies. It enables low-overhead mechanisms for response policy specification and dynamic response based on the suspect and incident context.

(3) We propose a distributed strategy to synchronize action executions in a neighborhood without significant communication overhead. Using a single timer to manage the action executions in a neighborhood supports minimization of redundant actions and load distribution. This design enhances the simplicity and scalability of the system.

(4) We propose an alternative daemon selection technique that allows network administrators to trade memory and communication overhead reducing potentially redundant and conflicting actions.

(5) We evaluate the performance of Kinesis through extensive TOSSIM simulations. We run simulations for various application and network layer incidents involving single and concurrent/multiple attackers. The results demonstrate that Kinesis always keeps data loss rate and transmission delays close to those of a typical attack free WSN. Kinesis also achieves redundant action optimization, load balancing among the neighbors, and energy efficiency.

(6) We port Kinesis to a real-world testbed of 37 TelosB motes and run experiments for (i) data loss, (ii) selective forwarding, and (iii) sinkhole attacks. Kinesis performance in the testbed are consistent to the simulation results.

8.1 Background and System Model

8.1.1 Case Studies

An important application of WSNs is intelligent surveillance for smart cities. Many traditional surveillance systems employ continuous video recording via CCTV cameras, with severe shortcomings in terms of excessive bandwidth, storage space, and constant human monitoring. Instead, the increasing need for safety in public and private areas is pushing the merge of physical and cyber worlds into smart surveillance systems [196–199]. Real-world deployments of such systems employ wireless sensors – able to detect movements, changes in temperature or light, vibrations, and more

– to activate the video recording, and independently and automatically alert a base station about intrusions.

As a concrete case study, Libelium [200] – a company focused on IoT, Machine2Machine, and Smart Cities solutions – has deployed several WSNs for intelligent surveillance and perimeter access control. Their sensors notify of intrusions over a low-bandwidth medium in self-organizing topologies, and trigger video recording to be streamed over a 3G network. As additional case study, SmartSantander [201] is one of the first holistic attempts at smart cities in Europe, deployed in the city of Santander, Spain. Among the various goals and solutions of this project, the add-on European project EAR-IT [202] aims at using a WSN with sound detection to use "intelligent acoustics" for detecting intrusions and danger situations, and notifying the authorities.

In these intelligent surveillance deployments, an attacker might compromise sensor nodes and/or tamper with the communications in order to drop the alert packets about his intrusion and cover his tracks. This can be easily identified as a concrete threat for several different kinds of facilities that employ intelligent surveillance systems, from corporate facilities, to government buildings, to homes. The employment of Kinesis in all these scenarios would ensure a quick response to data losses in the WSN. Moreover, as we show in our evaluation, the use of Kinesis in a WSN incurs a very small performance overhead, which would ensure to maintain the original application deployed on the nodes functional and responsive for this critical real-time task.

8.1.2 Network Model

We consider a multi-hop WSN, consisting of a number of sensor nodes and a base station (BS) that collects data from the network. A node is assumed to have more than one neighbor node which can monitor its behaviors. The BS is secure and has a secure mechanism to broadcast authentic messages and to disseminate code updates

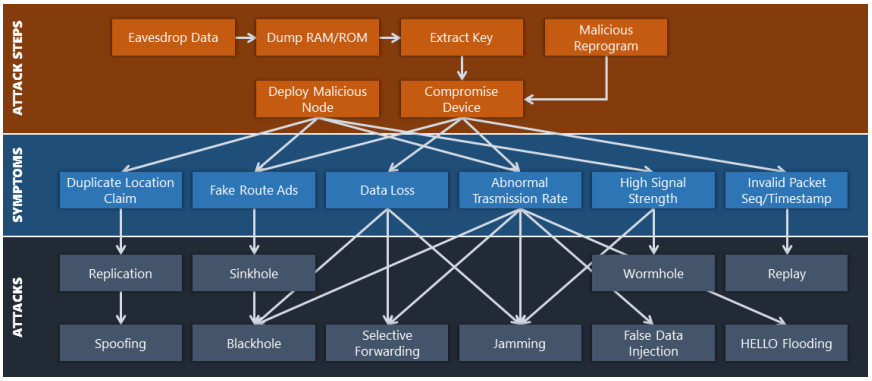


Figure 8.1. Attack graph

in the network. Sensor nodes are stationary after deployment, but routing paths may change over time, e.g., due to node failure. Once after the deployment, the BS assigns each node u a unique nodeID and a cryptographic key K_u. Each node also shares a pairwise key $K_{u,k}$ with each neighbor k and a group key K_g with all the neighbors.

8.1.3 Threat Model

We consider the BS as trusted, but any other node may be malicious. We assume a majority of honest nodes in a neighborhood. The WSN maintains the standard layered architecture of protocol stack which enables typical as well as WSN specific attacks to these layers. The attacks are directed to impair the following resources: (i) network, (ii) control and data message, (iii) sensor device resources, e.g. memory, power, etc. Below, we discuss these attacks with respect to the target resources.

Communication Network: *Jamming* disrupts a sub-network or even the entire network. Attacks at the link layer include purposely introduced collisions, resource exhaustion, and unfairness in medium access.

Messages: In a WSN, all the nodes act as routers. Hence, an attacker may spoof, alter, or replay routing messages to disrupt network traffic through creating routing loops, changing routes, attracting or repelling traffic from selected nodes, increasing latency, etc. Examples include *sinkhole*, *selective forwarding*, *blackhole*, *wormhole*

attack. Additional attacks, like false data injection and delayed forwarding, may be conducted to degrade data quality and utility.

Sensor Devices: To keep sensor networks economically viable, sensor devices come without tamper-resistant packaging, which adds the risk of physical attacks, e.g., physical capture, tampering, etc. An adversary can extract the secrets stored on captured sensors' chip and exploit software vulnerabilities. The adversary can also clone the captured sensors and place them into network at chosen locations (*replication* attack). Once these replicas gain the trust of others, they can launch a variety of insider attacks. ID spoofing, e.g. *Sybil* attack, poses threat by enabling a malicious node to present multiple false identities to the network.

8.1.4 Intrusion Detection System (IDS)

A number of IDSes [27–29] have been proposed for WSNs that cooperatively detect intrusions. Due to the broadcast nature of wireless channels, overhearing is a natural phenomenon in WSNs. Neighboring nodes overhear transmissions from each other, even if they are not the intended recipients [203]. Utilizing this fact, Marti et al. [29] introduce the *watchdog* mechanism by which a node identifies a misbehaving neighbor node by observing the neighbor behaviors. Such a node is termed *watchdog monitor* (a.k.a monitor). Each monitor observes its neighbors, collects audit data, and then performs behavioral analysis for each of them to detect any suspicious activities. The intrusions are cooperatively detected by the monitors based on their analyses, and a set of pre-defined inference rules. The relationships between the symptoms used by the IDSes and the various attacks are shown in Figure 8.1.

8.1.5 State Information and Notation

We now introduce the information used by Kinesis throughout its execution, together with some of the notation that we will use in the remainder of this Chapter.

Each node maintains a set of state information representing its current knowledge of the network and its security level. Each node u maintains a list of its direct neighbors $neigh(u)$ and link quality information $L(u, w)$ with each neighbor $w \in neigh(u)$. Also, u retains a per-neighbor *sliding window* w_k of size W to record a history of neighbor behavior observations, used by u to update the security estimation and state of the neighbors.

8.2 Architecture Overview

In Kinesis, each monitor hosts a distributed IDS and the Kinesis system. Through the IDS, a monitor observes neighbor behaviors, detects suspicious incidents in the neighborhood, and notifies Kinesis for automated response action. However, as we see in Section 8.4.2, Kinesis depends on the IDS only for the notifications on good/bad behaviors which is the basic functionality of an IDS. Hence, the specific design of the IDS is **out of the scope** of our work.

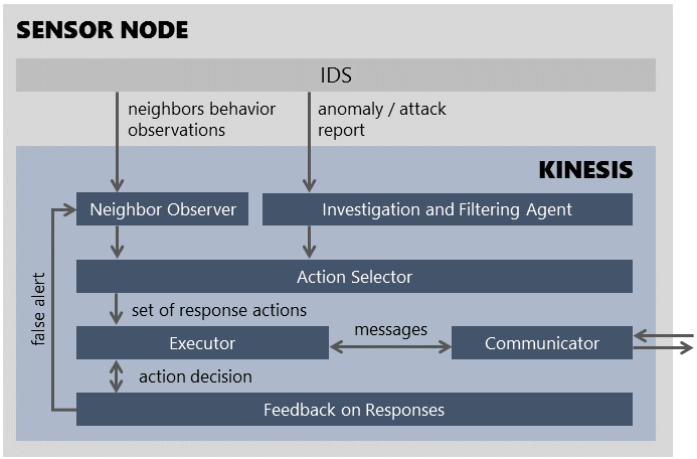


Figure 8.2. Overview of the Kinesis architecture

Figure 8.2 shows the architecture of Kinesis. The background process *Neighbor Observer*, with the help of IDS observations, records recent behaviors for each monitored neighbor and periodically updates the neighbor's security status based on this

history. Upon detecting an incident, the IDS reports to Kinesis the possible anomaly/attack, suspect node(s), and alert confidence for each reported anomaly/attack. A *Diagnosis and Filtering Agent* serves as the first entry point for such reports. This component, designed as an extensible pipeline, is responsible for ensuring an appropriate confidence in the IDS's reports, as well as for further disambiguation of reports concerning anomalies/attacks that may have similar symptoms but very different root causes, as discussed more in details in Section 8.3. The *Action Selector* then performs the *security assessment* of the suspect node based on the *alert confidence*, the *suspect behavior history*, and the *incident impact*. Based on the security assessment, the action(s) to be executed are selected dynamically from the response policy matched on the incident. Because of our incident-based approach, Kinesis can handle unknown attacks based on the anomalous behaviors they manifest.

Given a set of response action(s), the *Executor* triggers and executes the actions. A monitor competes to be the next *daemon* (i.e. the one to take the response action) by setting an action timer inversely proportional to its *action effectiveness* and takes the action when the timer fires. Note that actions such as *log* and *analyze* are executed by each node independently, whereas for actions like *retransmit_data*, redundant actions by the neighbors should be minimized. In the latter case, upon hearing an action taken by a monitor, other monitors in the neighborhood stop their action timers to refrain themselves from taking any further action for that incident. Any communication related to response actions or with the BS is handled by the *Communicator* module.

8.3 Diagnosis and Filtering of Adverse Events

IDSes typically associate a confidence value with each incident reported, in order to indicate the likelihood of its occurrence. Sometimes, however, the IDS might not provide a built-in confidence value, or such value might be very low due to the indecisiveness of the IDS about an incident. Moreover, traditionally IDS systems are able to detect incidents such as packet drops or data modifications, but not

the underlying causes of such incidents. Reporting incidents to Kinesis with good accuracy and a clear understanding of their root causes is critical to the selection of an effective response action.

The *Diagnosis and Filtering Agent* component of Kinesis implements an extensible filter pipeline (see Figure 8.3) that supports Kinesis in better diagnosing reported incidents, raising the confidence level of the notified adverse event, and reducing the number of possible root causes of such events. This way, when the notification reaches the IRS, the system is able to carry out a significantly more informed, data-driven decision, leading to a more effective action.

Figure 8.3. Overview of diagnosis and filtering pipeline prototype design

The first pipeline filter aims at back-filling the potentially missing confidence data in the incident reports. The confidence value is useful in selecting a response action, as it provides an estimate of the detection effectiveness of the IDS, as well as a suggestion about the expected severity of the response action. However, if the IDS does not provide a built-in confidence value, this first filter in the pipeline computes the confidence value R_C for the incident report as follows:

(i) For *Anomalies*, we consider $R_C = 1$. This is reasonable since watchdog monitors can correctly identify a failure or misbehaving event [29].

(ii) For *Attacks*, R_C is computed as a *false alarm rate* based on the past performance of the IDS about successfully detecting attacks. Thus, R_C is computed as:

$$R_C = \frac{\#\ of\ true\ attacks}{\#\ of\ attacks\ reported}$$

Details of how Kinesis gets feedback about false alerts are discussed in Section 8.4.5.

The second pipeline filter is the *Fine-Grained Analysis (FGA)* tool developed as part of our previous work [189]. This filter is vital in understanding the root causes that lead to an incident reported by the IDS. As an example, the IDS of a node might detect a packet dropping event, but it might not know whether the packet drop was caused by a compromised node performing a selecting forwarding attack, or by the introduction of interference – natural or malicious – disrupting the wireless transmissions. It is easy to see that, while both scenarios lead to packet loss, an effective action against the former would be the revocation of the compromised node, while an effective action against the latter would be the re-routing of traffic outside of the interference-affected area. Therefore, reporting incidents to Kinesis with good accuracy is critical to the selection of an effective response action. This strategy is particularly useful when several attacks share the same symptoms as shown in Figure 8.1. The FGA tool is able to distinguish between node- and link-related incidents, and can in the future be extended to discriminate among more kinds of incidents. The tool profiles the normal conditions for each link with the neighbors by measuring the link's received signal strength indicator (RSSI), link quality indicator (LQI), and the packet reception rate (PRR). The RSSI represents the signal strength between two nodes, measured in negative dBm (typically ranging in $[-100dBm, -45dBm]$), with higher values representing stronger signals; the LQI measures the quality of the signal; the PRR is the packet reception rate. In ideal situations, high RSSI values indicate a strong signal link and reveal better quality with high LQI and PRR values. However, the presence of interference induces noticeable changes in these values. Our fine-grained analysis is driven to understand the components of each link profile along with the neighborhood of each node. Initially after the network deployment, the stable profile parameters ($RSSI_{norm}, LQI_{norm}, PRR_{norm}$) for each link are saved at both the end-nodes. Upon the occurrence of an incident whose symptoms could be attributed to both node- and link-related issues, the FGA tool compares the current link profile with the stable parameters, and is able to determine the exact root cause for the reported incident.

Table 8.1.
Response policy language

<rules> ::= 'Begin' <rule-list> 'End'
<rule-list> ::= <rule> <rule-list> | <rule>
<rule> ::= 'on' <incident> (<condition> <action-list>)+
<incident> ::= <anomaly> | <attack>
<anomaly> ::= data_loss | data_alteration | data_replay | ...
<attack> ::= unknown | selective_forwarding | jamming | ...
<condition> ::= <condition>*|'if' <incident> 'then'
|'if' *severity*(<suspect>,<incident>) <op> (<value>|<range>) 'then'
<op> ::= '<' | '>' | '⇐' | '>=' | '==' | '!=' | 'IN'
<action-list> ::= <action>, <action-list> | <action>
<action> ::= <conservative-action> (<suspect>)*
|<moderate-action> (<suspect>)*
|<aggressive-action> (<suspect>)*
<aggressive-action> :: = *revoke* | *reauthenticate* | *rekey* | ...
<moderate-action> ::= *retransmit_data* | *trigger_data_authentication* | ...
<conservative-action> ::= *nop* | *analyze* | *alert* | ...
<suspect> ::= <digit>+ | <literal> (<literal>*<digit>*)*
<range> ::= ('['|'(') <value>–<value> (')'|']')
<value> ::= <digit> | <digit>+. <digit>+
<digit> ::= ['0'-'9']
<literal> ::= ['A'-'Z"a'-'z']

8.4 The Response Policy Language and Engine

In this section, we present the Kinesis policy language, and how Kinesis selects and executes appropriate response actions to incidents and anomalies.

8.4.1 Policy Language

A response policy is defined on an **incident** and specifies **actions** for different **security** estimations, based on various *condition*s on the incident and the suspect.

The response policies are specified as a set of rules, expressed with the grammar in Table 8.1. The terms within quotes ' ' are static tokens and the *italics* represent

functions. The main construct of the language is <rule> which defines the response policy corresponding to an attack or anomaly.

Through a detailed analysis of the various attacks in WSNs and corresponding recovery actions, we have identified a comprehensive set of response actions, listed in Table 8.3. The actions are categorized into three classes:

- *Conservative*: Low severity actions that may help a monitor in more precise attack detection or in not executing erroneous responses, but cannot prevent or recover from attacks.

- *Moderate*: Actions intended to preserve the WSN services under failures or attacks.

- *Aggressive*: High severity responses executed to recover from an attack and to prevent further malicious attempts. These actions may be executed by local sensors or may require help from the BS to execute them.

Note that the policy language of Kinesis allows for the use of generic symptoms as triggers, e.g. "`data_loss`", as well as anomalous behaviors detected by the IDS but not identified as a specific attack, i.e. "`unknown`". This lets Kinesis take action even in face of attacks that were not entirely diagnosed or detected, but that still require action to maintain the network functionality. The chain-like structure of our policies, moreover, ensures that the rules will be checked sequentially, reacting to more specific attacks when possible, and falling back onto general action rules as "safety nets" for the WSN. While this might result in a less accurate response action to an unknown attack, it ensures that some corrective measure is taken in all cases.

An example policy for *data_alteration* incident is shown in Table 8.2. Here, $nodeID$ refers to the suspect node identifier.

Table 8.2.
Response policy example

on 'data_alteration' if *severity*(data_alteration, nodeID) $\Leftarrow$ 0.3 then *retransmit_data* if *severity*(data_alteration, nodeID) IN (0.3,0.6] then *retransmit_data*, *trigger_route_change* if *severity*(data_alteration, nodeID) $>$ 0.6 then *revoke* nodeID

Table 8.3.
Taxonomy of response actions

Actions	**Descriptions**
CONSERVATIVE: Low Severity	
nop	No actions to take
log, *analyze*	Record auxiliary information and analyze
alert	Notify the suspect node(s) or other neighbors/the BS about the misbehavior
MODERATE: Medium Severity	
discard_data	Prevent forwarding false data
retransmit_data	Retransmits cached data
trigger_reauthentication	Re-authenticate the suspicious node
trigger_route_change	Change route and notify others
trigger_multipath_routing	Route data through multiple paths
suspend	Temporarily block the suspect node
AGGRESSIVE: High Severity	
revoke	Black list/block the convicted node
re-program	Re-program the malicious node
re-key	Re-key the (sub) network
flood_alerts	Flood alert messages in the network

8.4.2 Policy Matching and Response Selection

Since response policies are defined specific to incidents, it is straightforward to match the policy for an incident in Kinesis. However, the action to execute is selected dynamically from the action set specified by the matched policy, based on the *security*

assessment of the suspect. This strategy ensures that Kinesis takes the most effective action at any incident.

The security assessment of a node is quantified by a *Security Index (SI)*. In Kinesis, a monitor continuously updates per-neighbor security state records based on its observations of the neighbor behaviors. The SI of a neighbor is also updated on each observation. If a neighbor shows legitimate behavior, its SI is updated based on the behavior observations only. Otherwise, if an incident is reported (i.e. a misbehavior is observed), SI is updated based on three factors:

(i) Incident Confidence: The confidence with which the incident is detected, denoted by a *Confidence Index (CI)*.

(ii) Incident Impact: A numeric value of the impact of the incident on the WSN, denoted by an *Impact Index (II)*.

(iii) Neighbor behavior history: The continuous behavior observations and security state of the neighbor, reflecting how much the monitor believes the suspect node.

In what follows, we discuss how Kinesis computes these indices and then selects the response action based on the *SI*.

Confidence Index (CI)

At the end of the Diagnosis and Filtering pipeline execution, the report of an incident will always have a confidence value associated to it. We utilize it to select a response action since it measures how effective the IDS is in detecting an incident and how severe the response should be. The confidence value for the reported incident is therefore used as value for the CI.

Impact Index (II)

The *II* estimates the overall impact of an anomaly/attack and implies the urgency and severity of the response action. Despite extensive work on vulnerability scoring in enterprise networks [204], little attention has been paid to WSNs. A few mathematical risk models for WSNs have been proposed [205], but they do not provide a complete framework considering the WSN specific practical concerns. In this work, we propose a simple mechanism to estimate the impact of an incident.

Table 8.4 lists the consequences of incidents to the WSN services. Based on the priority of the WSN, the BS assigns static scores to the impacts and configures the nodes with the incident-impact mapping and impact scores. On receiving a report of incident x, Kinesis computes the incident impact as:

$$I^k(x) = \frac{\sum_{j=0}^{n} impact_x^k[j] \times r^k[j]}{\sum_{j=0}^{n} r^k[j]} \tag{8.1}$$

where k is the type of impact, n is the total number of *k-type* impacts, $impact_x^k$ is an *n-length* array of *k-type* impacts for incident x where $impact_x^k[j] = 1$ means that the incident has *j-th* impact, and r^k is an array of impact scores associated with the *k-type* impacts. Using Eq. 8.1, Kinesis computes the *Data Impact (I^d), Network Impact (I^n), Node Impact (I^s)* of the incident and then the II as follows:

$$II(x) = \beta_d \times I^d(x) + \beta_n \times I^n(x) + \beta_s \times I^s(x) \tag{8.2}$$

where, the coefficients $\beta_d, \beta_n, \beta_s \geq 0$ are real numbers such that $\beta_d + \beta_n + \beta_s = 1$. Note that if the network administrator does not change the WSN priorities, the *Impact Index*es are **static** and need to be calculated only **once** after deployment.

Table 8.4.
Possible impacts of WSN anomalies and attacks

Data Impact	Data delay
	Data unavailability
	Data alteration
	Data falsification
Network Impact	Network unavailability
	Network disruption
	Path unavailability
Node Impact	Node unavailability
	Node misbehavior
	Node malfunction

Neighbor Behavior Observations

The neighbor behaviors help a monitor assess how vulnerable the neighbor is and how likely it is that the neighbor is going to attack. Hence, we consider the behavior observations of the suspect node while determining the severity of the response action. Usually IDSes maintain the behavior history and trust scores [206]. However, to conform with IDSes without such facilities, Kinesis includes a mechanism for recording information about the neighbor behaviors and utilizing ing this information in computing security score and state.

To justify the accuracy of the response action, we utilize the history of neighbor behaviors rather than the latest single behavior. Kinesis maintains a per-neighbor *sliding window* w_k of size W to keep track of the neighbor's most recent W behaviors. When the IDS notifies about a behavior of neighbor k, Kinesis pushes out the oldest behavior from w_k and stores the recent one. We consider two types of behaviors:

- *Service Behavior*: How trustworthy a neighbor node is in providing WSN services, e.g., in-time packet forwarding.

- *IPRS Behavior*: How efficient and honest the neighbor is in taking required and desired actions.

Security Index and State Update

A monitor u computes the SI for each neighbor $k \in neigh(u)$ on each behavior observation for k and updates the security state accordingly. A node is estimated to be in five possible states: (i) Fresh, (ii) Suspicious, (iii) Secure, (iv) Malicious, and (v) Revoked. Figure 8.4 shows the security state transition diagram. After the network deployment, a monitor assigns to all its neighbors the *Fresh* state with $SI = 0$. For a pre-specified amount of time t_f, a neighbor is considered to be in *Fresh* state while its SI is updated based on behavior observations according to Eq. 8.3. The significance of *Fresh* state is that a neighbor is given the *benefit-of-doubt* while being in this state. Although the SI of a suspect node in *Fresh* state affects the response selection, no aggressive action is taken against the node, i.e., the node is not revoked, reprogrammed, etc. After a time t_f, the neighbor moves to *Suspicious* or *Secure* state based on its SI. A node in the *Suspicious* state moves to the *Secure* state if its SI decreases due to legitimate behaviors. On the contrary, if a node in the *Suspicious* state continues its anomalous behavior, its SI goes above a pre-defined threshold σ_2 and the node moves to the *Malicious* state. When a neighbor moves to the *Malicious* state, the monitor initiates an aggressive action against the node. A neighbor node can also be revoked anytime due to the monitor's own decision or action initiated by neighboring monitors. In this case, the monitor enlists the suspect node as *Revoked* and discards further request/data from the node.

We compute the SI of a neighbor k with two auxiliary functions $f(x)$ and $g(SI)$, where $f(x)$ computes the *severity* of an incident x and $g(SI)$ returns a coefficient based on the current SI and security state of k.

$$g(SI) = \begin{cases} 1 & ; SI \leq \sigma_1 \text{ i.e } k \text{ is } \textit{Fresh/Secure} \\ 1.5 & ; \sigma_1 \leq SI \leq \sigma_2 \text{ i.e } k \text{ is } \textit{Suspicious} \\ 2 & ; SI > \sigma_2 \text{ i.e. } k \text{ is } \textit{Malicious} \end{cases}$$

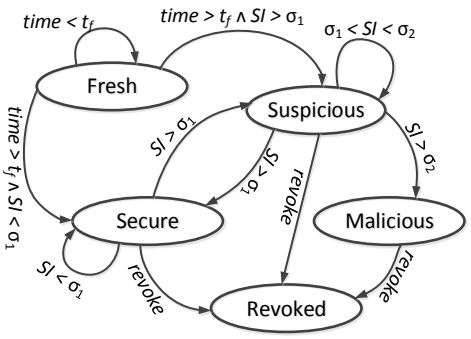


Figure 8.4. Security state diagram of a monitored node

$$f(x) = \begin{cases} 0 & ; x \text{ is good behavior} \\ min(CI \times II(x) \times g(SI), 1) & ; \text{otherwise} \end{cases}$$

On each i-th behavior observation for neighbor k, its SI is computed by a monitor as

$$SI = \begin{cases} \frac{\sum_{j=1}^{i} f(w_k[j])}{i} & , \text{if } i \leq W \\ \frac{\sum_{j=1}^{W} f(w_k[j]) - f(w_k[0]))}{W} & , \text{if } i > W \end{cases} \tag{8.3}$$

8.4.3 Response Computation and Optimization

If the IDS reports a single incident corresponding to an incident, Kinesis computes the *SI*, matches the response policy, and selects the *SI* based action(s) from the matched policy. When multiple incidents are reported, we follow the same procedure to select the action(s) for each reported incident and compute the final action set as a union of these actions. However, each individual action set may be inclusive, overlapping, inconsistent with respect to the other sets. Moreover, before considering new action(s) for execution, we should also check inconsistencies with the on-line actions. To resolve this issue for a limited resource system, we introduce the action precedence graph.

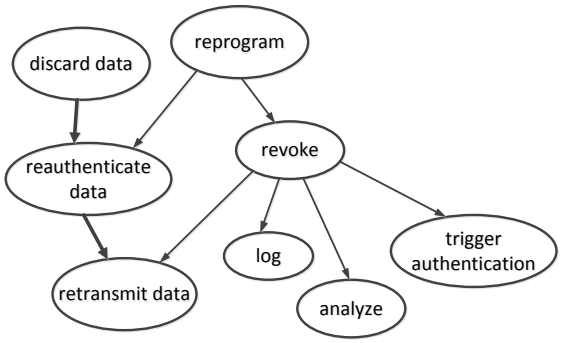


Figure 8.5. Example of an action precedence graph

The **Action precedence graph** (APG) is a directed graph which describes the precedence relationship between actions in terms of their effectiveness. In such graph, (i) each node a_i represents an action, (ii) an edge $a_i \rightarrow a_j$ denotes that the parent action a_i invalidates the child action a_j, and (iii) a *black edge* $a_i \Rightarrow a_j$ denotes that a_i and a_j are contradictory actions and on conflict, a_i is executed. Thus the execution of action a_i invalidates all of its successors, and a_j *not reachable by* a_i means that they are independent actions. Two actions a_i, a_j conflict if one can reach the other only through a path of black edges. An example APG is shown in Figure 8.5 where: the *reprogram* action overrules all of its successors; {*log, analyze*} are independent of each other; and {*retransmit_data, reauthenticate_data, discard_data*} conflict. We assume that the **BS pre-configures** the nodes with all possible response actions and the precedence relationships between them.

By utilizing the APG, Algo. 5 computes the equivalence, independence, intersection, and coverage relationships between two action sets. To compute the optimized action set from n different action sets $\{A_1, A_2, \ldots, A_n\}$ (each specific to an individual incident), Kinesis runs a recursive algorithm initialized with $O_1 = A_1$ and then computing $O_i = cors(O_{i-1}, A_i)$ for $i = 2, 3, \ldots, n$.

ALGORITHM 5: : cors() - computation of optimized response set

Input: Response sets A = $\{a_i\}$, B = $\{b_i\}$
Output: Optimized response set O
if $A = B$ **then**
    O ← A ; // A is *equivalent* to B
else if $\forall a_i, \exists b_j,\ b_j \rightarrow a_i$ **then**
    O ← B ; // B *covers* A
else if $\forall a_i, \forall b_i,\ a_i \Rightarrow b_j$ *or Vice-versa* **then**
    O ← A (or B) ; // A *contradicts* B
else if $\exists a_i, \exists b_j,\ a_i \rightarrow b_j$ **then**
    O ← A ∪ (A\B) ; // A *intersects* B
else
    O ← A ∪ B ; // A is *independent* to B
end

8.4.4 Execution of a Response Action

The response action executions are fully distributed. The low/medium severity actions are executed by the monitors solely based on their own decisions. The high severity actions against convicted nodes require consensus among the neighborhood monitors. In the latter case, a selected monitor node (*daemon*) broadcasts a message asking the decisions of other monitors, performs a *majority voting* on the collected replies, and then executes the agreed upon action. Some aggressive actions, such as *reprogram* or *rekey*, cannot be completed at the sensors. In such a scenario, the *daemon* node notifies the BS with an authenticated report and the BS then performs the action. In addition, even though some actions, like *retransmit_data, alert_others*, can be executed upon a monitor's own decision, they require interactions with other nodes. In all these cases, **a monitor has to initiate** the action and take over all the related responsibilities. Kinesis dynamically selects the most competent node as the *daemon* to ensure the action effectiveness and to avoid the same node doing all the work all the time.

Selection of the Daemon

A node is selected as the *daemon* via a self-organized competition among neighboring monitors. The novelty of our scheme is that we do not need any message exchange or special time synchronization among neighbors to manage the action executions. Each node in a neighborhood competes independently through a locally managed back-off timer, called *action timer*. The timer value of a node u depends on the *action effectiveness*, $AE(u)$, of the node, which is estimated locally based on: (i) neighborhood size, (ii) one-hop link qualities, and (iii) time since last action. Intuitively, if a node has more neighbors with good link qualities, it can interact with more monitors and help minimize redundant actions. Again, if the node is idle for a long time, it should take the action to ensure load distribution in the neighborhood. Thus, the $AE(u)$ is computed as follows:

$$AE(u) \propto c_1 \cdot t_l + c_2 \cdot \sum_{\substack{k \in neigh(u) \\ k \in neigh(s)}} L(u,k) \tag{8.4}$$

Here, c_1, c_2 are real numbers, $neigh(u)$, $neigh(s)$ denote the neighbors of u and the suspect node, respectively, $L(u,k)$ is the link quality between u and the monitor k, and t_l is the *time since last action* by u. The higher the $AE(u)$, the more effective u's action is. u joins the competition to be next daemon by setting the timer value inversely proportional to $AE(u)$. We add to this value a small random time r, in order to ensure that, even when the action effectiveness values at two monitors are close – that is, when the *load balancing factor* (i.e. time since last action) is the same in both of them and the *link qualities* with the neighbors do not make a big difference – their action timers still have different values. This additional small random factor helps in making sure that the action timers of two monitors do not fire simultaneously, and therefore only one monitor is selected as daemon.

$$ActionTimer(u) \propto \frac{1}{AE(u)} + r \tag{8.5}$$

Thus, a node with better AE has lower back-off period and wins the *daemon* selection competition. When the action involves a transmission and a neighbor k overhears it, the node stops its running timer to avoid any redundant action for the same incident and updates its t_l and AE value. Kinesis could allow redundant actions with a goal to enhance the system reliability. For example, suppose that node u drops data packets and one of its monitors retransmits the dropped packets. If another attacker v in the data flow path drops the retransmitted packets, then redundant transmissions by multiple neighbors of u may help mitigate data loss. However when v drops data, its neighboring monitors retransmit the data, which invalidates the necessity of redundant actions by u's neighbors. Our experimental results also support the design of minimizing redundant actions as we see very low data loss rate in the presence of multiple attackers. We investigate redundant actions further in Section 8.5.

Consensus Among the Monitors

To execute high severity actions, the monitors consult with each other and decide an action based on **majority voting**. After selecting a response action, the *daemon* node broadcasts an authenticated *status_req_msg* in the neighborhood. The message contains the (i) detected attack, (ii) the suspect node, (iii) the response decision, and (iv) a Message Authentication Code (MAC) computed on the data using the group key K_g.

Upon receiving the message, each neighboring monitor replies with an authenticated *status_reply_msg*, containing the local response decision. The *daemon* node computes and broadcasts again the majority voting result. Based on the voting decision, the *daemon* may execute the agreed upon action or notify the BS with an authenticated report to trigger the action. The neighboring monitors also observe

each other to check whether they abide by the voting decision and otherwise records a *bad* behavior for the misbehaving node.

8.4.5 Response Feedback

The majority voting decision gives a feedback to the monitors about their accuracy in terms of detecting an incident and selecting the actions. If the severity of the agreed upon action is lower than the locally determined action at a node, it implies a false alarm and decreases the confidence of the monitor. Every monitor node keeps the records of its false alarms and updates its *CI*. Note that we do not consider false negatives here. Response feedback may also help assess the effectiveness of the taken action for an incident. However, we do not investigate this direction in this work.

8.5 Redundant and Conflicting Actions

The ability of Kinesis of taking a single action per incident, as it would be ideal, mainly depends on the topology of the network, among other factors. As our experimental evaluation in Section 8.7.3 shows, occasionally, multiple monitors simultaneously undertake an action for the same incident. The concurrent execution of more than one action for the same adverse event can impact the battery life of the monitor nodes as well as, in extreme cases, lead to inconsistencies in the network if the redundant actions performed are both of high severity and conflicting.

In scenarios in which the network administrator has control over the topology of the WSN, he or she can adjust the placement of the nodes to address the issue of redundant and conflicting actions. Under such scenarios, the techniques used by Kinesis to select the daemons discussed so far are optimal with respect to the security goal as well as the resource constraints of the nodes. However, to deal with cases in which the network administrator does not have much control over the topology of the network, we introduce a daemon selection technique that allows one to trade some additional memory and communication overhead for energy saving and consistency

assurance in face of redundant and conflicting actions. It is worth noting that the security goal of Kinesis remains anyway unchanged. The desired trade-off between the efficiency of the various resources (memory, energy, communication channel), together with the specific requirements of the individual WSN application, will let the network administrator establish which daemon selection technique is more appropriate on a per-case basis.

8.5.1 Conflicting Actions Analysis

First of all, we analyze all the actions supported by Kinesis in pairs, to determine which pairs of actions conflict with each other. Formally, we define two actions as *conflicting* if they obstruct each other in reaching the security goal of Kinesis, which is to minimize data losses.

Table 8.5 summarizes the results of our analysis, with the 'X's marking conflicts. For example, the actions `discard_data` and `retransmit_data` aim at opposite end results. If the packet contains false data, then retransmitting it will propagate false information throughout the network; conversely, if the data is true, discarding the packet will not let valuable information reach the base station. So if the two actions are performed concurrently by different monitor nodes, the conflict might make it harder for Kinesis to reach its security goal. As shown in the table, however, there are only 5 instances of conflicting actions. In the remainder of this section, we discuss how the alternative daemon selection technique helps Kinesis in mitigating the potential issues of redundant and conflicting actions.

8.5.2 Redundancy Motivating Scenario

To understand the reasons behind the phenomenon of redundant actions, consider the network portion in Figure 8.6, with the edges denoting direct communication between two nodes. Node 18 is an attacker node, while Nodes 7, 8, 9, 13, 14 and 29 are the monitors. When Node 18 drops a packet, all the monitors start their action

Table 8.5.
Analysis of potentially conflicting response actions

	nop	log	alert	discard_data	retransmit_data	reauthenticate	change_route	route_multipath	suspend	revoke	re-program	re-key	flood_alerts
nop	-												
log		-											
alert			-										
discard_data				-	×			×					
retransmit_data				×	-				×				
reauthenticate						-							
change_route							-						
route_multipath				×				-					
suspend					×				-	×	×		
revoke									×	-			
re-program									×		-		
re-key												-	
flood_alerts													-

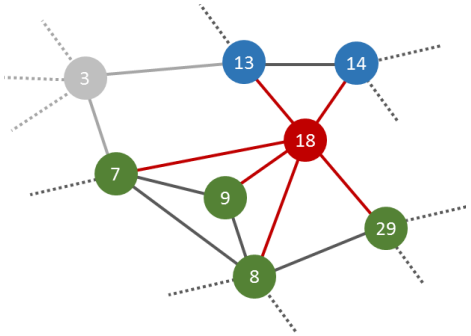


Figure 8.6. A segment of the attacker's neighborhood

timers. In this topology, two different kinds of situations can lead to redundant actions:

1. **"Hidden Node":** When, for example, Node 7 wins the action timer competition, it retransmits the dropped data and Nodes 8 and 9 stop their timers upon

overhearing the action. However, Node 29 does not have a link to Node 7, thus it is a "hidden node" with respect to the action taken by Node 7, and will also take action when its timer fires. [1]

2. **Disconnected Neighborhood Portions:** Nodes 13 and 14 have no direct connections with any of the nodes 7, 8, 9, 29. Therefore, their portion of the neighborhood of Node 18 is disconnected from the other portion. In this case, no matter which node in one portion of the neighborhood of Node 18 takes action, the other portion will never overhear the action. The only solution is therefore introducing an action-reporting mechanism among the portions. Node 3, even though not directly interested in the action since not directly connected to the malicious Node 18, could relay the notification that an action has been taken from one portion of the neighborhood to the other. In case of high-severity actions, it would be very important to propagate the information about an undertaken action to the rest of the interested neighbors.

The alternate daemon selection technique presented in this section provides a solution to both scenarios, at the cost of additional memory and communication overhead. At a high level, such solution is composed by two parts, the first addressing the "hidden node" problem with an analysis of the connectivity between monitors for any specific incident, and the second addressing the disconnected portions problem with an automatic reporting mechanisms that still minimizes the communication overhead.

8.5.3 2-hop Knowledge

At the base of any solution against redundant actions, there is the need for each node to be more aware of the topology of its neighborhood, as well as the connectivity of its other neighbors, in order to make better informed decisions. Each node, when meeting its direct neighbors at network startup, also acquires the list of its neighbors.

[1] This kind of redundancy is not a sole problem of Kinesis, but of any overhearing-based solution.

That is, node u will store the list $neigh(w)$ for each 1-hop neighbor $w \in neigh(u)$. This is defined as *2-hop knowledge*. Every node therefore is aware of the topology of its network up to 2 hops away. Given an incidence matrix M representation of the graph of the entire wireless sensor network, a node u knows all and only columns v for which $M[u, v] = 1$.

8.5.4 Connectivity Advantage

Let us assume a node u wants to start the action timer after its direct neighbor b caused an adverse event – e.g. dropped a packet. Let us define the direct neighbors of node u that are also direct neighbors of b as the local action set of u with respect to b, denoted as $LAS(u, b)$. Nodes in such set are the only nodes in the direct neighborhood of u that will be setting an action timer for the adverse event caused by b.

$$LAS(u, b) = neigh(u) \cap neigh(b) \setminus b$$

Each node, knowing its 2-hop neighbors, can determine how well each of its direct neighbors is connected to the neighbors of the malicious node b. We define the ratio of neighbors of b that a node v is directly connected to as *connectivity ratio* $CR(v, b)$, computed as:

$$CR(v, b) = |neigh(b) \cap neigh(v)|/(|neigh(b) \vdash 1)$$

In order to maximize the number of nodes that overhear an action undertaken and thus minimize the number of redundant actions, our solution ensures that the node that wins the action timer competition is the one connected to as many interested nodes as possible. For this reason, we introduce the *Connectivity Advantage* $Adv(u, b)$ of node u over the nodes in the local action set of nodes u and b. Before providing a formal definition, we informally introduce this concept. Intuitively, the Connectivity Advantage parameter determines how well a node u is connected to the other neighbors of a malicious node b, that are the monitors for every incident caused by the

latter. Since node u is taking part to the monitoring of node b, this implies that u is a direct neighbor of b, and therefore knows the set of direct neighbors of node b – whether they are directly connected to u or not – thanks to the 2-hop knowledge that node b provided to node u at network startup. We formally define the Connectivity Advantage of node u over the nodes in the local action set $LAS(u, b)$ as a parameterized factor ranging between α and $1/\alpha$ that will increase (or decrease) the action effectiveness of u[2].

When a node u has to set its action timer for an adverse event caused by a node b, it needs to calculate its own Connectivity Advantage. Therefore, it calculates the connectivity ratio CR for each one of the nodes in the local action set $LAS(u, b)$, including itself. Once all the connectivity ratios are calculated, the node can normalize them in the $[0, 1]$ range as normalized connectivity ratios $NCR(u, b)$:

$$NCR(u,b) = \frac{CR(u,b) - \min\limits_{w \in LAS(u,b)} CR(w,b)}{\max\limits_{wLAS(u,b)} CR(w,b) - \min\limits_{wLAS(u,b)} CR(w,b)}$$

Node u will finally use its own $NCR(u, b)$ value to determine its Connectivity Advantage as a value in the aforementioned range $[\alpha, 1/\alpha]$ as:

$$Adv(u,b) = (1/\alpha - \alpha)(NCR(u,b)) + \alpha$$

It is easy to see that a normalized connectivity ratio equal to 1 will result in a Connectivity Advantage of $1/\alpha$, while vice versa a normalized connectivity ratio equal to 0 will result in a Connectivity Advantage of α. Thus, this function fulfills the requirements of increasing or decreasing the delay in the action timer depending on the better or worse connectivity of a node to the direct neighbors of a malicious node, in order to reach as many neighbors as possible with an action upon winning the competition and therefore to limit redundant actions.

[2]Remember that a smaller value for the action timer means more chances to win the competition, and therefore the best advantage will result in a factor of $1/\alpha$, while the worst will be a factor of α.

Based on $Adv(u, b)$, the equations in this alternative technique for action effectiveness and action timer at node u for an adverse event cause by its neighbor b are, respectively, as:

$$AE(u) \propto c_1 \cdot t_l + c_2 \cdot \frac{1}{Adv(u, b)} \tag{8.6}$$

$$ActionTimer(u, b) \propto \frac{1}{AE(u)}$$

Here, c_1, c_2 are real numbers, and t_l is the time since last action by u.

It is evident that a node might not be directly connected to all the other nodes in the local action set. This means that each node has only partial information about such nodes, and this will affect the values that are calculated locally at each node. This is a purposeful design choice that limits the information exchanged among the nodes to a 2-hop knowledge in order to enable advanced calculations such as the Connectivity Advantage, while limiting the memory and storage overhead of the nodes. The algorithm is guaranteed to provide a local optimum that can be efficiently computed without the need for interaction with the other nodes (apart from the initial neighbor list exchange at network bootstrap).

8.5.5 Proofs of Action

Depending on the particular topology of the malicious node's neighborhood, there are cases in which favoring the node best connected to all the other neighbors of the malicious node – by considering its Connectivity Advantage – cannot completely solve the issue of redundant actions. This happens when no single node in the neighborhood of interest is directly connected to all the others, or also in the more general case when such neighborhood is divided in multiple, disconnected sub-portions. Here, some nodes are said to be *"hidden"* with respect to the particular undertaken action(s). To deal with such scenarios, the solution provided by Kinesis includes a mechanism for automatic notification to the interested parties, via *Proofs of Action*.

While it is important to make sure that high-severity actions are reported to all the interested nodes immediately – before more redundant and potentially conflicting high-severity actions are undertaken by unaware nodes in a different part of the neighborhood – for non-conflicting redundant actions, the main concern is the unnecessary consumption of energy. Since adding extra rounds of communication would effectively defeat the purpose, Kinesis intelligently differentiates between these two security goals and only activates the notification mechanism in case of high-severity actions. In case of non-conflicting (i.e. low-severity) redundant actions, it is more energy effective to simply let multiple actions coexist.

We now describe how this mechanism helps in reducing the number of redundant actions. Let us analyze again the scenario of a node b causing an adverse event. Several monitors will witness the event and will start their action timers (considering also their Connectivity Advantage). Suppose that the timer of node v fires first, then node v undertakes action A. All and only the direct neighbors of v will overhear that action, and will stop their action timers. In order to solve the hidden node problem, each one of these direct neighbors will then leverage the 2-hop knowledge in order to check if any of its direct neighbor is hidden with respect to action A by node v, proceeding as follows. Without loss of generality, let u be one such node. From the point of view of u, the nodes that are hidden with respect to action A originated by node v against the malicious node b are all those nodes w such that b is one of their direct neighbors, but v is not. More formally, such hidden nodes compose the set defined as follows:

$$hidden(v, b) = \{w : b \in neigh(w) \wedge v \notin neigh(w)\}$$

If the set $hidden(v)$ is not empty, and action A has potential conflicts with any other action (as per Table 8.5), then node u will generate and broadcast an authenticated proof of action to let the hidden nodes know about the action and prevent them from undertaking redundant and potentially conflicting actions.

Since this algorithm is fully distributed, the information will keep propagating hop by hop to all the interested nodes, i.e. those that witnessed the adverse event but could not overhear the action taken and might attempt to start one on their own. It is important to mention that the number of these nodes is always bounded by the number of direct neighbors of the malicious node minus 1, and depending on the topologies might be much lower than this upper bound.

Algorithm 6 shows the pseudocode for the solution provided by Kinesis for managing redundant actions, composed of both the computation of the Connectivity Advantage, its use in setting the action timer upon the detection of an attack, and the use of proofs of action to propagate information to the hidden nodes. The algorithm has been optimized in order to only keep the bare minimum variables necessary for the computation, thus lowering the memory requirements. For example, instead of keeping in memory all the connectivity ratio values, it only computes in place the minimum and maximum of their range in order to later on normalize only the value for the node that is performing the calculation itself.

8.6 Implementation and Configuration

We implement Kinesis in TinyOS 2.x. We adapt the Skipjack encryption based CBC-MAC implementation in TinySec [207] for TinyOS 2.x to compute a 4-byte MAC while majority voting. The implementation is lightweight since it takes 0.38 ms for Mica2 motes [207] and would take less time in the TelosB platform since TelosB has higher processing capability than Mica2 mote.

According to the policy language defined in Section 8.4.1, policy rules are implemented as *switch-case* based on incident. This strategy optimizes the implementation. Security state thresholds (σ_1, σ_2) are used to specify the severities in policies. To compute σ_1, σ_2, we average over all the incident impacts, measure the SI with this average impact for various attack rates, and select the values based on the tolerance

ALGORITHM 6: : Pseudocode for the solution against redundant actions (at node u)

```
on adverse_event(b) do
    my_ae ← computeAE(); // omitted as trivial
    my_adv ← computeAdv(b);
    setActionTimer(my_ae, my_adv);
endon
function computeCR(v, b) {
    common ← (neigh(b) ∩ neigh(v)).size();
    common / (neigh(b).size() - 1);
}
function computeCR(v, b) {
    common ← (neigh(b) ∩ neigh(v)).size();
    return common / (neigh(b).size() - 1);
}
function computeAdv(b) {
    for v ∈ neigh(b) do
        if v ∈ my_neighbors then
            conn ← computeCR(v, b);
            if conn < min_cr then  min_cr ← conn ;
            if conn > max_cr then  max_cr ← conn ;
        end
    end
    my_ncr ← (computeCR(self, b) - min_cr) / (max_cr - min_cr);
    my_adv ← (1/alpha - alpha) * my_ncr + alpha;
    return my_adv
}
on overhear_proof_of_action(p) do
    overhear_action(p.timestamp, p.A, p.originator, p.malicious);
endon
on overhear_action(timestamp, A, originator, malicious) do
    cancelActionTimer();
    if alreadySeen(timestamp, A, originator, malicious) ∨ ! isHighSeverity(A) then
        return;
    end
    hidden ← ∅;
    for w ∈ my_neighbors do
        if malicious ∈ neigh(w) ∧ originator ∉ neigh(w) then
            hidden ← hidden ∪ w;
        end
    end
    if hidden ≠ ∅ then
        p ← generate_proof_of_action(A, originator, malicious);
        broadcast(p);
    end
endon
```

to attack rates. The particular set of response actions used to specify the policies are determined based on the security goals of the WSN application.

As Kinesis configuration, the WSN administrator configures the sensors with the incident-impact mappings, impact scores, and the real coefficients. The *data*, *network*, or *node* impacts of an incident do not vary across different WSNs, hence the incident-impact mappings are static. On the contrary, how severely an incident affects the WSN services may well depend on the network application. Thus, the impact scores and β coefficients, used to compute the II, should be set by the administrator according to the application requirements. However, we assume that these configuration parameters are changed infrequently over the network lifetime.

8.7 Simulation Results

In this section, we present simulation results to show the performance of Kinesis under various network settings.

8.7.1 Simulation Setup

For simulation, we use the TinyOS simulator TOSSIM. The network topologies are generated with symmetric links. As a routing protocol, we use the standard *Collection Tree Protocol* (CTP). In the experiments, we consider the following application and network layer anomalies and attacks: (i) data loss, (ii) data alteration, (iii) selective forwarding, and (iv) sinkhole attack. The policies considered for these incidents are shown in Table 8.6. To detect incidents, we implement a simple watchdog monitor based IDS in TinyOS 2.x.

To configure Kinesis, we assign equal weight to the real coefficients in Eq. 8.2, i.e., $\beta_d = 0.34$, $\beta_n = 0.33$, $\beta_s = 0.33$. The size of the per-neighbor *sliding window*, W, is set to 100. In Section 8.4.4, we have stated how we determine the values of σ_1, σ_2. A data source periodically sends out data every 2 seconds. In each simulation run, the

results are averaged over 3,000 data transmissions. Unless otherwise stated, we use the above default values in all simulation runs.

Table 8.6.
Considered response policies

on 'data_alteration' if *severity*(data_alteration, nodeID) IN (0,0.2] then *retransmit_data* if *severity*(data_alteration, nodeID) IN (0.2,0.4] then *change_route*, *retransmit_data* if *severity*(data_alteration, nodeID) > 0.4 then *retransmit_data*, *revoke* nodeID
on 'data_loss' if *severity*(data_loss, nodeID) IN (0,0.2] then *retransmit_data* if *severity*(data_loss, nodeID) IN (0.2,0.4] then *change_route*, *retransmit_data* if *severity*(data_loss, nodeID) > 0.4 then *retransmit_data*, *revoke* nodeID
on 'selective forwarding' *retransmit_data*, *revoke* nodeID
on 'inconsistent_etx' if *severity*(inconsistent_etx, nodeID) IN (0,0.4] then *NOP* if *severity*(inconsistent_etx, nodeID) > 0.4 then *revoke* nodeID
on 'sinkhole' *revoke* nodeID

8.7.2 Performance Metrics

The metrics considered to evaluate Kinesis are:

1. **Effectiveness:** Since our goal is to minimize the impact of data, network, etc. failure, we show the effectiveness of Kinesis from two aspects:

   (a) *Data Loss Rate at the BS*: The frequency with which the BS experiences the effect of an incident i.e. the rate of reception failures at the BS. In this context, we compare the performance of our system with (i) **an attack free typical sensor** environment, and (ii) an **under-attack network**

to show that Kinesis can get back the WSN into a normally operating environment, even under anomalies or attacks.

(b) *Average Data Transmission Delay*: On average, the amount of time a packet takes to reach the BS since its transmission by the source. Here, we compare the performance of Kinesis with an **attack free** scenario.

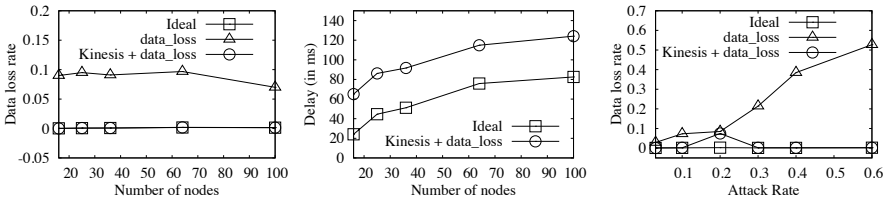
2. **Optimization of Redundant Actions:** The *average number of actions taken per incident* by the monitors in a neighborhood. We also measure the *rate of redundant actions per incident* as a ratio of the number of monitors taking response actions to the number of monitors that detected the incident. They justify our *action timer* design based distributed scheme to trigger the response actions.

3. **Load Balance:** How evenly the response action executions are distributed in the neighborhood. This is indicated by the standard deviation among the number of actions taken by the monitors in a neighborhood.

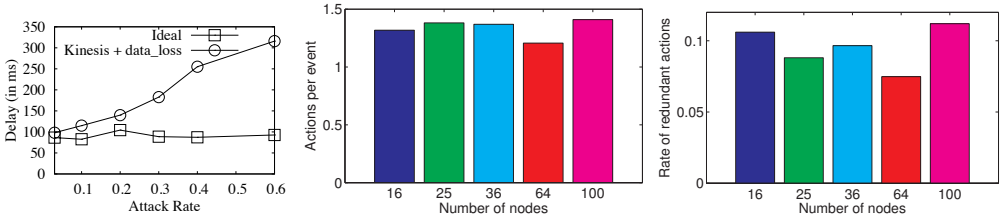
4. **Energy Consumption:** The sum of energy usage by all the nodes when Kinesis along with an IDS is in operation.

8.7.3 Grid Network Experiments

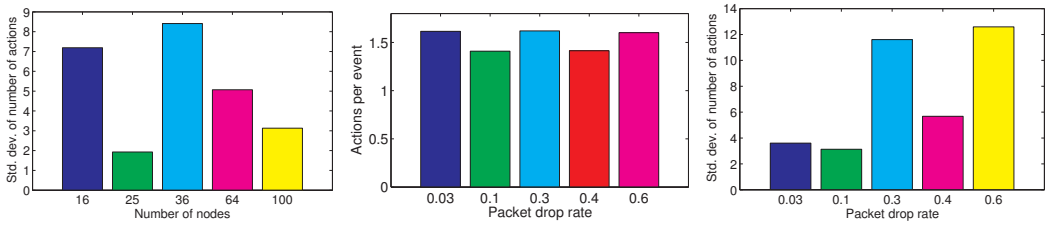
We place 16 to 100 nodes in grid topologies of dimensions from 4×4 to 10×10, respectively. The nodes are spaced 1.5 meter apart. For each network, a data source and an attacker are randomly selected and the results are averaged over 10 runs. The attack rate is set to 0.1. For concurrent attacks, we place a second attacker both in the same and different neighborhood than the first one. Both attackers are equally likely to make an attack.



(a) Data reception failure rate at the BS (b) Average data transmission delay (c) Data reception failure rate at the BS



(d) Average data transmission delay (e) Average number of actions per *data_loss* incident (f) Rate of redundant actions per *data_loss* incident



(g) Load balance among neighbor monitors (h) Average number of actions per *data_loss* incident (i) Load balance among neighbor monitors

Figure 8.7. Kinesis performance for *data_loss* of rate 0.1 in grid networks of various sizes and for various attack rates in a 10×10 grid network.

Single Attack

First, we show the performance of Kinesis in case of a single incident (anomaly/attack) in the network.

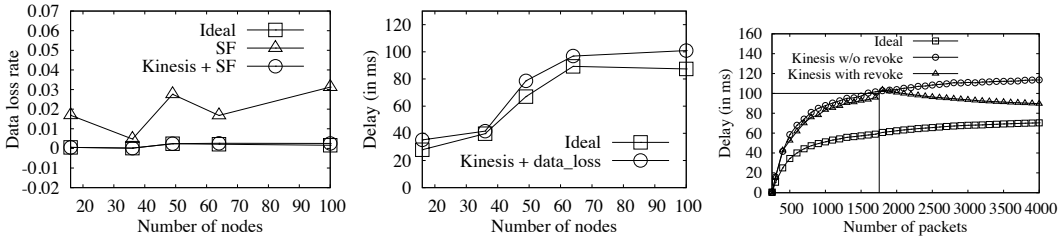
***data_loss* Incident**: In this case, a node may be faulty or malicious and drops data packets intermittently instead of forwarding them to the BS. Figure 8.7(a), 8.7(b), 8.7(e)-8.7(g) show the performance of Kinesis under *data_loss* incidents in WSNs of sizes from 16 to 100. As shown in Figure 8.7(a), Kinesis reduces the data loss rate of a network under attack from [0.073, 0.103] to ~ 0.002, which is similar to the natural

data loss rate (~ 0.0018) in a network without attack. It proves the effectiveness of Kinesis both in small and large networks.

Figure 8.7(b) shows the linearly increasing trend in average transmission latencies with network sizes. However, the average latency Kinesis adds due to action execution is almost invariant ([39.03, 41.607] ms) in different networks. The delay incurred by Kinesis is mostly due to the *action timer*. According to Eq. (8.4.4) and (8.5), the *action timer* value depends on the number of neighbors and the link qualities with them. In the experiments, neighborhood sizes vary from 3 to 5 in different networks and the link quality values lie in [0.8, 0.976]. The combined effect of neighborhood size and link qualities makes the *action timer* values almost invariant in different networks. Thus, the increasing trend in transmission delays is mainly due to the increase in routing path length with network sizes.

Figure 8.7(e) shows that Kinesis is not always able to take a single action per incident as in ideal case. Occasionally, it triggered as much as 1.4 actions per incident on average. However the rate of redundant actions, as shown in Figure 8.7(f), is bounded by 0.11. The small standard deviation ($[1.93, 8.41]$) in the number of actions by neighboring monitors, as shown in Figure 8.7(g), indicates the high success of Kinesis in load balancing.

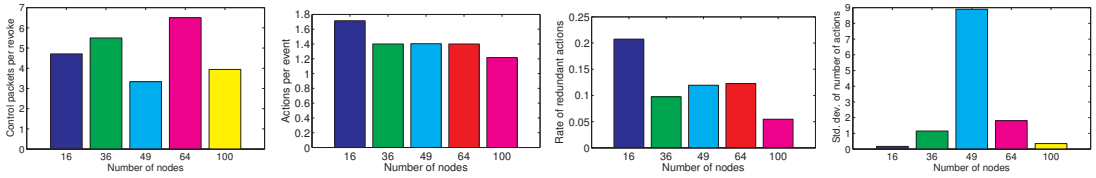
To further analyze the scalability of Kinesis, we measure its performance under various attack rates in a 100-node network and show in Figure 8.7(c), 8.7(d), 8.7(h), 8.7(i) how well Kinesis survives, even for very high attack rates. As expected and consistent to earlier results, Kinesis counteracts the data loss attacks and gets the network back to normal operating condition. Figure 8.7(c) shows that Kinesis reduces the data loss rate of a network under attack from [0.02, 0.52] to $\sim$0.0001, which proves its effectiveness and scalability, even under higher attack rates. Figure 8.7(d) reveals the linearly increasing trend in average transmission latencies with higher rate attacks. Even average latencies introduced by Kinesis with varying attack rates are negligible ([12,223] ms).



(a) Data reception failure rate at the BS

(b) Average data transmission delay

(c) Avg. transmission delay over packets



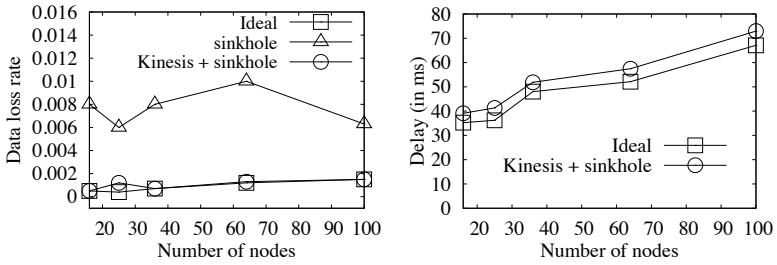
(d) Average number of control message exchanges in a neighborhood on *revoke*

(e) Average number of actions per incident

(f) Rate of redundant actions per incident

(g) Load balance between neighbor monitors

Figure 8.8. Kinesis performance for *selective_forwarding* (SF) attacks in grid networks of various sizes



(a) Data reception failure rate at the BS

(b) Average data transmission delay

Figure 8.9. Kinesis performance for *sinkhole* attack

Figure 8.7(h) shows that the average number of actions per incident is ∼1.5. The action redundancy per incident is bounded by 0.16. However, both numbers are almost invariant with respect to attack rates. This is because the number of actions depends on the link quality among the neighbors and the differences in their action timer values. Figure 8.7(i) shows a small standard deviation in the number of actions

taken by the neighbors, which indicates the effectiveness of the distributed scheme of Kinesis in triggering action executions.

***data_alteration* Attack**: In this attack, a malicious node selectively modifies the data value in a data packet before forwarding it to the BS. We run simulations for *data_alteration* attacks and find similar trends in the results as in *data_loss* incidents. Later on, we show the performance of Kinesis for concurrent incidents of *data_loss* + *data_alteration*, hence we do not report the graphs here.

***selective_forwarding* Attack**: In a selective forwarding attack, the monitor nodes initially observe data loss by the attacker and hence retransmit the dropped data. Once they detect a selective forwarding attack is occurring, the *daemon* issues a *state_req_msg* to the neighborhood. The neighboring monitors reply with their own action decision about the suspect in a *status_reply_msg*. Based on the majority voting decision from the replies, the daemon possibly issues a revocation request to the BS. The BS then disseminates a *revoke* command to the network, upon receiving which all the nodes exclude the attacker from the routing path.

Figure 8.8 reports the performance of Kinesis under selective forwarding attacks in networks of various sizes. In a selective forwarding attack, no matter whether the attacker is revoked from the network or not, Kinesis retransmits the packet dropped by the attacker. Hence, Kinesis reduces the data loss rate of a network under attack to that of a network without attack. Figure 8.8(a) supports the claim by showing that the natural data loss rate and the loss rate of a network under attack with Kinesis enabled are almost equal.

Figure 8.8(b) shows an interesting and significantly different trend in transmission delays with Kinesis under *selective_forwarding* attack. In this case, the average transmission delays are much lower compared to that of *data_loss* incidents and quite close to the natural data transmission delays. To analyze the performance better, we show the average transmission delays over time in Figure 8.8(c). Initially when the monitors do not detect the selective forwarding attack yet but only observe data losses, they retransmit dropped packets and thus add latencies to data transmissions.

After the revocation of the attacker at packet 1755, there is no attack and hence no delay is incurred due to response execution.

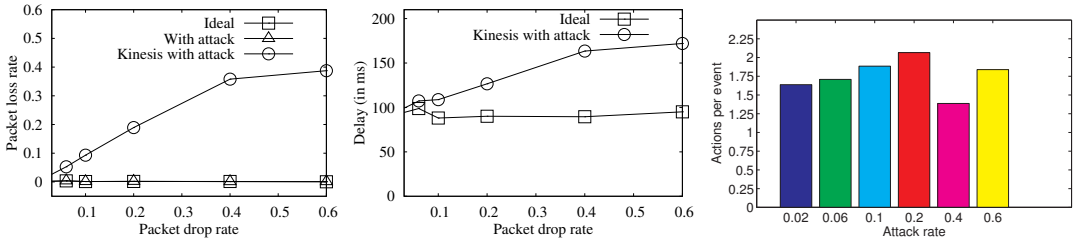
Figure 8.8(d) shows the average number of control messages (*state_req_msg* + *status_reply_msg*) exchanged in a neighborhood for majority voting. The *state_req_msg* is of 27 bytes and *state_reply_msg* is of 35 bytes. The number of control messages per majority voting is $\leq$6.2 packets. However, it is proportional to the neighborhood size and thus does not vary with network sizes unless the number of neighbors varies.

Figure 8.8(e) - 8.8(g) show that the average number of actions, action redundancy and load distribution measurements are consistent with the earlier experiments and can be explained in a similar way.

For the *selective_forwarding* attacks, the monitors always agreed on the decision to *revoke* the suspect node. The average time to perform the majority voting and execute the decided action is $\sim$ 96.4 ms, most of which is contributed by the action timer value.

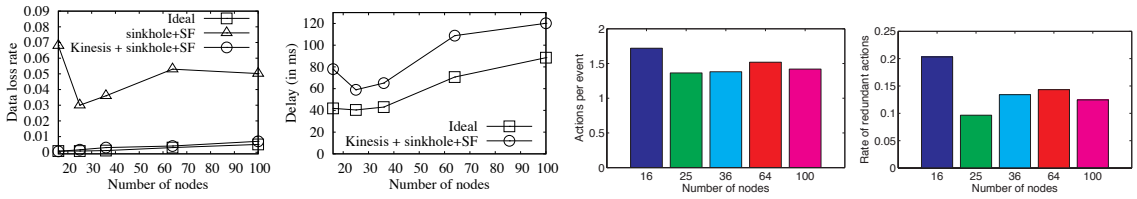
***sinkhole* Attack**: To simulate this attack, we modify the CTP protocol to enable the attacker advertising low cost routing path through it. Once the attacker attracts all the data in the neighborhood, it drops data at a rate of 0.2.

In Kinesis, a monitor suspects a potential *sinkhole* attack upon hearing an inconsistent path cost advertisement, which results in an update of the SI for the suspect but NOP as a response. During the subsequent packet drop observations, the monitors retransmit the dropped data and eventually revoke the malicious node when the attack is confirmed. Figure 8.9(a) shows that Kinesis reduces the data loss rate to $\sim$ 0.0015. At the same time, it keeps the transmission delays closer to natural latency, as shown in Figure 8.9(b), due to the quick revocation of the attacker node. Note that the *sinkhole* attack often created routing loop causing as high as 3.5% data loss. By revoking the attacker, Kinesis made the WSN stable again.



(a) Data reception failure rate at BS (b) Average data transmission delay (c) Average number of actions per incident

Figure 8.10. Kinesis performance for *data_loss+data_alteration* incidents with various rates in a 10×10 grid network



(a) Data reception failure rate at the BS (b) Average data transmission delay (c) Average number of actions per incident (d) Rate of redundant actions per incident

Figure 8.11. Kinesis performance for *sinkhole + SF* attacks in grid networks of various sizes

Concurrent Attacks

In the concurrent attack experiments, we consider two cases, two simultaneous but independent attackers, and two colluding attackers.

- In case of two concurrent but independent attackers, we consider an attacker causing *data_loss* and the other conducting *data_alteration* at various rates in a 10×10 grid WSN.

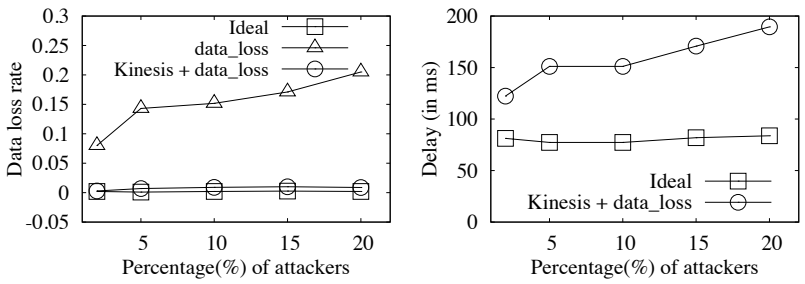
  As we see in Figure 8.10, Kinesis shows behaviors consistent with the single attack scenario, in all the aspects. Thus, Kinesis is effective under concurrent and high rate attackers.

- Next, we consider two colluding attackers performing *sinkhole* and *selective_forwarding* (SF) attack. When the *sinkhole* attacker is revoked, routing path

changes enable data routing through the *SF* attacker which then drops data at a rate of 0.5, and vice versa. Figure 8.11 shows how Kinesis performs in such scenario. The irregularity occurring when the number of nodes is equal to 16 is due to the temporary routing instability after revocations.

Varying the Number of Attackers

To further show the scalability of Kinesis, we present its performance in a multi-attacker environment. Here, we consider *data_loss* incidents and vary the number of attackers from 2% to as high as 20% of the total nodes in a 100-node network. Figure 8.12(a) shows that Kinesis still keeps the data loss rate lower than 0.009. Due to Kinesis, the average transmission latencies vary within [122.33,189.46] ms, as shown in Figure 8.12(b). The results are consistent to earlier results.



(a) Data loss rate at the BS (b) Average data transmission delay

Figure 8.12. Kinesis performance for *data_loss* for various % of attackers at rate 0.1 in a 10×10 grid network.

Energy Consumption

We measure and compare the aggregated energy consumption of the WSN under various incidents while Kinesis (as well as the IDS) is in operation and in an attack-free scenario where Kinesis is not deployed (baseline). For energy measurement, we consider *MICAz* platform and use PowerTOSSIM z [208] plugin. Due to the

scalability limit of PowerTOSSIMz, we consider a 6×6 grid WSN with one source and one attacker.

In a Kinesis enabled system, overhearing does not incur overhead since it is inherent in WSNs. In TinyOS, the radio stack requires a node to receive and process all the packets transmitted in a neighborhood to understand whether the packet is destined to it or not. TinyOS also exposes the Receiver [209] interface that allows one to perform actions upon overhearing a message in transit. Thus, the only energy overhead imposed to the nodes is due to the IDS and Kinesis operations. The results reported in Table 8.7 show that Kinesis system incurs only a maximum of 0.06% energy overhead.

Table 8.7.
Aggregated energy cost of the WSN without and with Kinesis + IDS

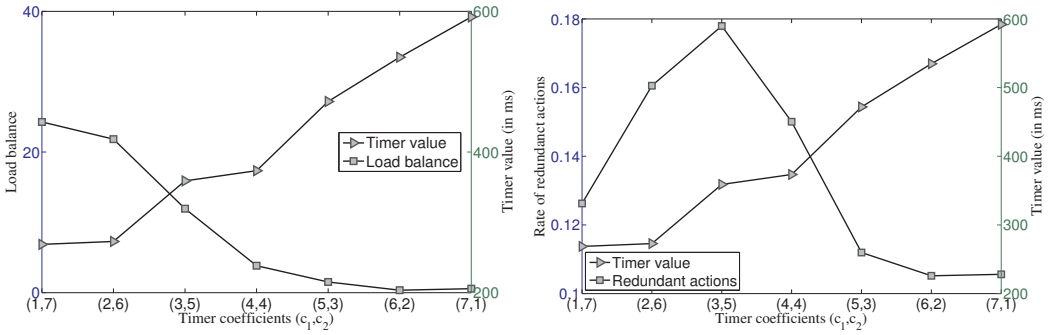
—	Baseline	Kinesis		
		data_loss	SF	sinkhole
Energy usage ($\times 10^7$ mJ)	1.320488	1.320482	1.321356	1.320480

The vast majority of the energy consumption of a WSN node is due to radio communications. Since in case of increasing attack rate – which for data loss events means higher number of dropped packets – the communication rate does not consequently increase, the energy consumption analysis results in Table 8.7 are representative of any attack rate from the point of view of a single Kinesis-equipped node.

Action Timer Configuration

Action timer design is crucial in Kinesis and its configuration impacts the performance with respect to redundant actions and load balance. Hence, we vary the coefficient factors (c_1, c_2) in Eq. 8.4.4 and analyze the impact of timer values on Kinesis performance. Since c_1, c_2 are weight coefficients, $c_1 + c_2$ should be bounded to optimize the timer value. If $c_1 + c_2$ is too small, the action timer fires frequently

which increases the number of actions. If $c_1 + c_2$ is too big, the latency increases. In our experiment, we fixed $c_1 + c_2$ to 8. Figure 8.13(a) shows that the optimum values of (c_1, c_2) in terms of load balance are near (3,5). In Figure 8.13(b) the optimum values are after (4.5, 3.5). To optimize both the action redundancy and load balance, (c_1, c_2) should be selected onwards (4.5, 3.5).



(a) Load balance vs. Action timer (b) Action redundancy vs. Action timer

Figure 8.13. Coefficient configuration for action timer

Benefits of Diagnosis

We analyze how the Diagnosis and Filtering pipeline enhances the capabilities of Kinesis to take the most appropriate action for an adverse event by knowing the root cause of the incident. To do so, we compare the performance of the network with Kinesis under the two situations in which the diagnosis features of the architecture are turned on and off. In particular, we analyze the scenario in which the IDS detects the loss of data packets and notifies Kinesis; however, such packet loss is caused, in turn, by selective forwarding attacks and by the introduction of interference/jamming by an external node not part of the network. In our evaluation scenario, only one attack at a time is active in the network; one malicious node is programmed to carry out a Selective Forwarding attack discarding packets with 20% probability, and the jamming attack is able to drop up to 30% of the traffic. The FGA tool part of the diagnosis pipeline is able to differentiate between selective forwarding and interference

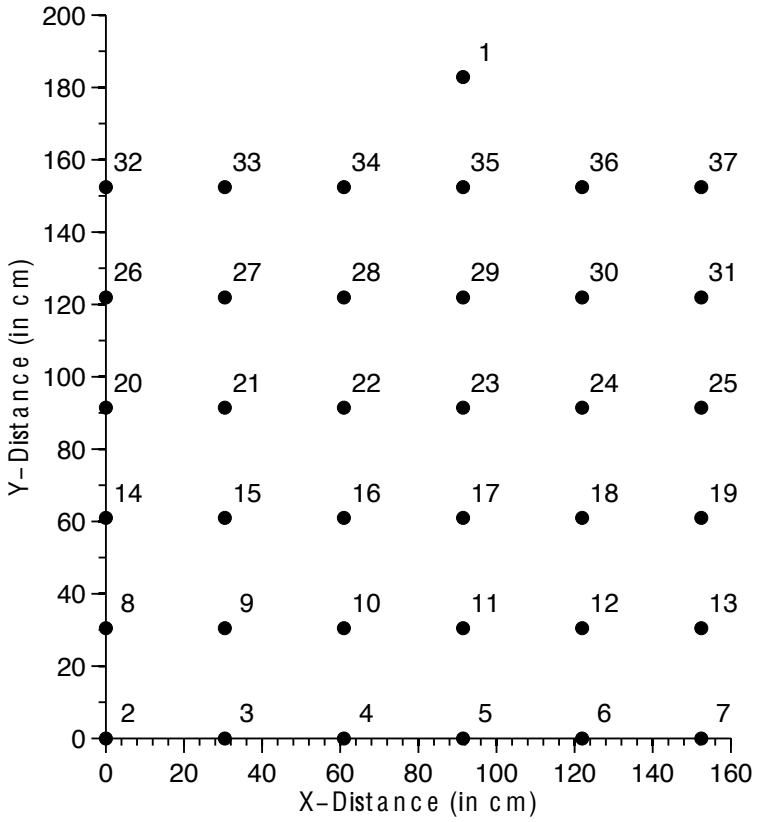


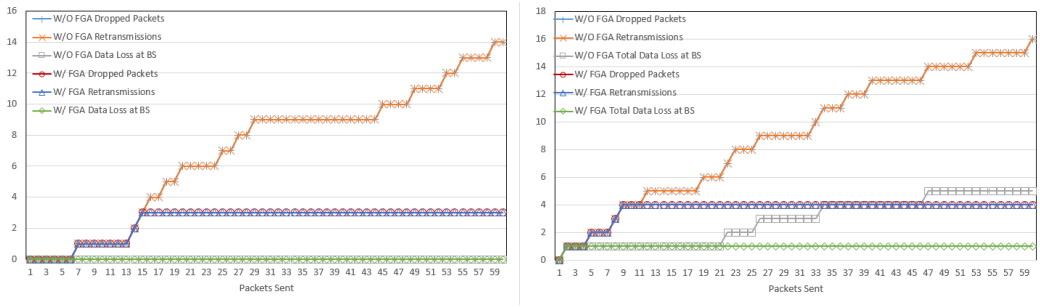
Figure 8.14. Node placement in an indoor 6×6 grid WSN

with an average accuracy of 92.5% [189]. We consider a simplified response policy, as shown in Table 8.8. Data loss is the general anomaly that the IDS reports to Kinesis; the diagnosis pipeline is then responsible to further differentiate between selective forwarding and interference attacks.

Figure 8.15 shows the results of our evaluation, plotting various measures of packet losses and retransmission over the number of packets generated and sent. In Figure 8.15(a) we can see the result with respect to the selective forwarding attack. With and without the presence of the FGA tool, there is no data loss at the BS, thanks to the retransmissions of Kinesis. However, without the FGA tool, the attack is not correctly identified and the malicious node is not revoked: this lets it continue to drop packets and requires Kinesis to keep on retransmitting the dropped packets, wasting resources. On the other hand, the accurate diagnosis by the FGA tool leads

Table 8.8.
Response policy for diagnosis scenarios

on 'data_loss' *retransmit_data*
on 'interference' if *severity*(interference) IN (0,0.2] then *retransmit_data* if *severity*(interference) > 0.2 then *retransmit_data*, *trigger_route_change*
on 'selective forwarding' *retransmit_data*, *suspend* nodeID



(a) Selective Forwarding attack (b) Interference attack

Figure 8.15. Benefits of the fine-grained analysis on response effectiveness

to an early revocation of the malicious node, and therefore the packet loss is halted much earlier (around packet number 15).

Figure 8.15(b) shows the results for the interference attack. This time, the data loss at the BS, even though very small, is non null. The reason is that some of the retransmitted packets are still dropped because of the interference in the most affected areas. Tthe plotted lines for "Retransmissions" in the figure denote an attempted retransmission, but not necessarily a successful one; for this reason, the graph shows a retransmission for each dropped packet, but a non-null data loss at the BS. When the FGA tool provides Kinesis with an accurate root cause for the packet loss incidents, Kinesis is able to trigger the most appropriate action, that is, re-routing the traffic

away from the affected area, and greatly limits the packet loss and the subsequent retransmissions by the daemons.

We find therefore that, both under the selective forwarding attack and the interference attack, the accuracy of the diagnosis pipeline in detecting the root cause of an incident greatly improves the effectiveness of the selected response actions, minimize the retransmission and save resources, and reduce potential data losses.

Alternate Daemon Selection and Redundant Actions

We compare the original daemon selection technique with the the alternate daemon selection technique presented in Section 8.5, to quantify how much the latter is able to limit the number of redundant actions. In our evaluation scenario, we vary the number of nodes in the network, and carry out a selective forwarding attack. We measure the number of actions per incident that the monitors undertake.

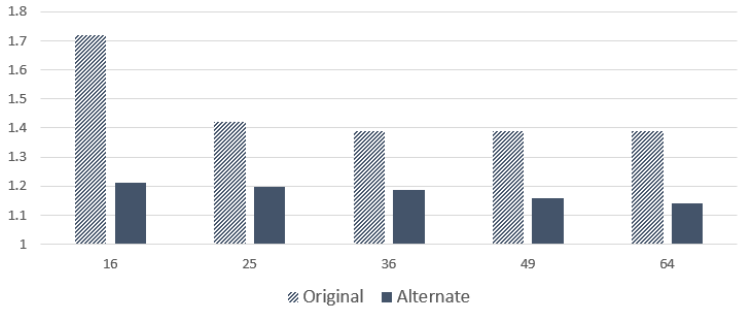


Figure 8.16. Comparison of the original and alternative daemons selection technique with respect to average number of actions per incident with varying number of nodes

Figure 8.16 shows the results of our evaluation. The alternate daemon selection technique leveraging the connectivity advantage is consistently able to reduce the number of actions per incident on all network sizes. In particular, the results average to 1.18 actions/incident, which indicates that with this strategy Kinesis is almost always able to carry out a single action per incident. These results confirm our

expectations that more information about the connectivity of the monitors in each individual neighborhood leads to a better informed decision in the selection of the daemon.

Proof of Action Overhead

In order to determine the communication overhead of Proofs of Action in terms of number of additional packets sent, we set up an evaluation scenario that stress-tests this feature by simulating exclusively actions that require proofs of actions (i.e. high severity or potentially conflicting actions, as described in Section 8.5.5) for different network sizes. For each number of nodes, we vary the nodes acting as data source and suspect node, and present averaged results. Specifically, we measure the number of proof of action packets sent per incident, as well as the coverage of the hidden nodes that are reached by the proofs of action and therefore became aware of the actions undertaken (thus avoiding starting conflicting actions). Figure 8.17 shows the results of our evaluation. For every network size, always 100% of the hidden nodes are reached. This result confirms the theoretical expectation that each hidden node should eventually be reached, possibly after more than one hop. We also observe that, on average, no more than about 3 packets need to be sent to reach all the hidden nodes. Note that for attackers with a small neighborhood, this quantity might include some redundant proof of actions sent by nodes at the opposite edges of the neighborhood, while for attackers with bigger neighborhoods, this number is more influenced by the number of propagation hops that a proof of action needs before reaching all the hidden nodes. For an increasing network size, the average size of the neighborhoods as well as the number of hidden nodes does not grow linearly, and therefore the number of proof of action packets does not increase linearly either. As an additional note, while this evaluation is a stress-test that triggers proofs of action for every incident, in normal scenarios many of the actions undertaken by Kinesis will not require this mechanism to be activated; the average number of additional

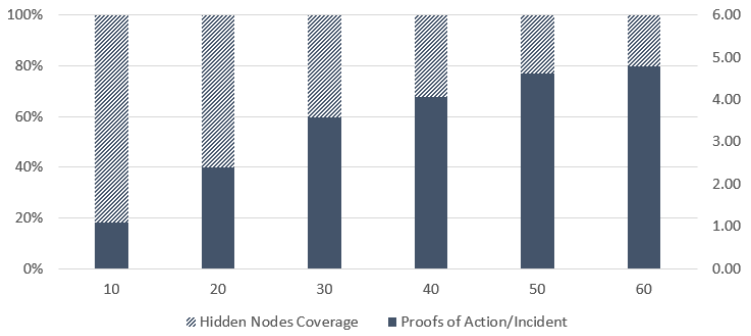


Figure 8.17. Evaluation of the proof of action overhead

packets over the lifetime of the WSN is therefore bound to be significantly low. We can see, then, that the mechanism of proofs of action has a quite low overhead, while still providing benefits in case of potentially conflicting actions.

Scalability

In the design of Kinesis, each node only maintains state about its direct neighbors. Therefore, the absolute scale of the network does not affect the practicality of the system; instead, the relevant scalability factor is the average neighborhood size, which however is physically bound to reasonably small sizes and entirely independent from the total scale of the WSN. Nevertheless, we evaluate the performance of Kinesis in a large-scale WSN of 529 nodes (a grid of 23x23 nodes) carrying out a selective forwarding attack with increasing attack rates (from 5% to 40%). Figure 8.18 shows the results. The baseline – with no attacks – is slightly lower than in smaller-scale networks, as routing becomes more complicated due to loop-induced queue overflows, mutual interference on multi-path duplications, no-ack drops, and more [210]. Nevertheless, across all attack rates, Kinesis maintains the network functional by guaranteeing a Packet Delivery Ratio comparable to that in absence of attacks (averaging at 90.5%) even in case of a very large network.

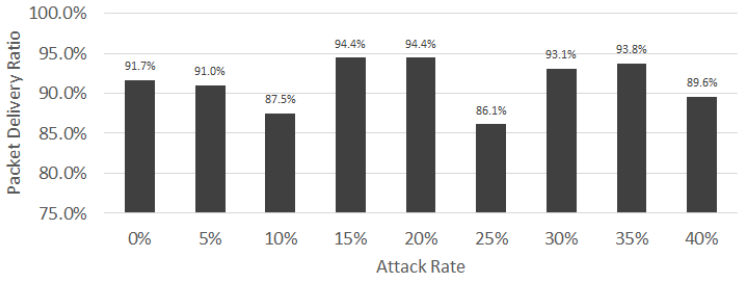


Figure 8.18. Evaluation of the scalability of Kinesis in a large-scale WSN with increasing attack rate.

8.8 Testbed Evaluation

We ported the Kinesis implementation to the TelosB platform and placed battery-powered TelosB motes in an indoor environment. Our motes have a 8 MHz TI MSP430 microcontroller, 2.4 GHz radio, 10 KB RAM, and 48 KB ROM. We ran experiments for *data_loss*, *selective_forwarding*, and *sinkhole* attacks and use the same metrics as in simulation for performance evaluation. The results of the experiments are averaged over 1500 packets.

8.8.1 Experimental Setup

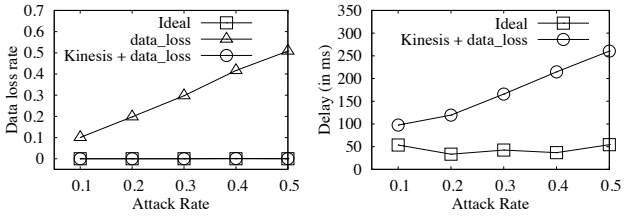
We build a 6×6 grid WSN, consisting of 36 TelosB motes, in a $160 \times 200\,cm^2$ indoor environment. Figure 8.14 shows the coordinates of the network nodes, labeled from 2 to 37. Node 10 is the source, which sends data every 1 second.

We controlled the transmission power of motes to ensure multi-hop communication. A special mote, labeled as node 1, is set to the root node. For performance analysis, the root collects statistics on the number of data and action packets transmitted, number of actions per incident, transmission delays and passes these data to a laptop through serial forwarder.

8.8.2 Kinesis Performance

Below, we present the testbed performance of Kinesis for *data_loss*, *selective_forwarding*, and *sinkhole* incidents. For the first two incidents, we set node 16 as the attacker.

***data_loss* Incident**: We evaluate the performance of Kinesis under various rates of *data_loss* incidents. Figure 8.19(a) shows that Kinesis reduces the data loss rate of the WSN under attack from [0.1, 0.51] to ≤ 0.0015, similar to the natural data loss rate. The average transmission delays when Kinesis is in operation vary within [97.5, 260.4] ms, as shown in Figure 8.19(b). Kinesis triggers on average [1.28, 1.97] actions per incident. Thus the testbed performance of Kinesis is consistent to that in simulations and justifies the simulation results.



(a) Data loss rate at the BS

(b) Average data transmission delay

Figure 8.19. Testbed performance of Kinesis for *data_loss* incidents of various rates in a 6×6 grid WSN.

***selective_forwarding* (SF) Attack**: Table 8.9 summarizes the performance of Kinesis under SF attack, where the attacker drops packets at a rate of 0.4 and is revoked at packet 604. Hence there is no attack and Kinesis actions afterwards, which keeps the average transmission delays much lower compared to that of *data_loss* incidents.

***sinkhole* Attack**: We conduct two sets of *sinkhole* attack experiments, setting two different nodes 21 and 22 as attackers. Once an attacker is able to attract surrounding data packets, it drops data at a rate of 0.2. The performance results of Kinesis are presented in Table 8.10.

Table 8.9.
Testbed performance of Kinesis on SF attack

	Ideal	SF	Kinesis + SF
Data loss rate	0.0008	0.064	0.0008
Avg. transmission delay (ms)	32.89	N/A	61.11
Avg. actions per incident	N/A	N/A	1.6875

Table 8.10.
Testbed performance of Kinesis on *sinkhole*

	Ideal		sinkhole		Kinesis + sinkhole	
	Exp 1	Exp 2	Exp 1	Exp 2	Exp 1	Exp 2
Data loss rate	0.011	0.086	0.015	0.20	0.011	0.086
Avg. transmission delay (ms)	71.17	113.03	N/A	N/A	75.27	177.36
Avg. actions per incident	N/A	N/A	N/A	N/A	1	1.604

The results of experiment 1 are quite similar to simulation results. The attacker is revoked at packet 158. It is to be noted that the sinkhole attack in this case created routing loops due to low cost path advertisements by the attacker and thus resulted in data loss. However, Kinesis took a quick response action to revoke the attacker, which brought back the routing stability and helped keep the data loss rate minimal.

In experiment 2, we see comparatively higher data reception failure and transmission delay at the BS. This is due to the routing instability created when Kinesis revoked the attacker at packet 376. Consequently, some packets were lost while a few others needed unusually longer time to reach the BS until a stable routing path was re-established.

Energy Consumption: Due to the difficulty of measuring energy directly on the sensor hardware [73], we adopt the energy model proposed by Polastre et al. [74] to estimate the energy cost in testbed. The energy cost of a node is estimated as a sum of energy usage due to sensing, transmission, and reception. The energy for a type of operation is computed by multiplying the battery voltage with the current draw and

time spent (according to the TelosB datasheet) for the operation. The aggregated energy cost of the WSN in case of baseline, *Kinesis+data_loss*, *Kinesis+SF* are 4232.86, 4447.44, 4467.46 mJ, respectively. Thus, Kinesis (along with the IDS) incurs a maximum of 5.5% energy overhead.

8.9 Security Analysis

In this section, we discuss various ways in which an attacker might try to circumvent the security mechanisms of Kinesis, and how our system deal with such situations.

Majority Voting. A set of colluding attackers may mislead the majority voting to decide on a wrong response action. If the attacker(s), replying with a low severity action, can affect the voting decision to be an action of lower severity than those reported by honest monitors, it also makes them detecting a false positive and lowering the monitor confidence. These attacks, however, will not succeed as we assume a majority of honest nodes in a neighborhood.

A solution to deal with such attacks on majority voting is to set higher weights on the local decisions of more trustworthy nodes. An alternative approach is to use complementary methods with Kinesis to detect such attacks. For example, we may use our previous work on lightweight provenance techniques that enables the BS to detect a data dropping attack and identify the misbehaving node, based on the data provenance, i.e., the identities of the source and routing nodes that processed or forwarded the data towards the BS [211]. In case a number of colluding attackers falsely report an honest node as a data dropping attacker and ask the BS to take aggressive action (e.g., *revoke*) against it, the BS will find an inconsistency between the reports of the colluding attackers and the data provenance. The reason is that, based on the provenance information, the BS will not be able to detect any data dropping attacks by the honest node. The BS may then conclude about a highly

probable collusion attack and respond accordingly. As part of future work, we will extend our current implementation by integrating such an approach.

Connectivity Advantage. The only drawback in the use of the Connectivity Advantage for the setting of the action timer is the impact that it will have on load balancing. In fact, it is easy to see that better connected nodes will typically win more action timer competitions and thus perform more actions, resulting in a quicker consumption of their energy. However, at the granularity level of neighborhoods, limiting redundant actions as much as possible will concurrently save energy for other nodes, thus balancing the global neighborhood energy consumption.

One may wonder whether a compromised node could collude with an attacker node and pretend to be a badly connected neighbor (when exchanging the neighbor lists at startup), therefore forcing the other nodes to always take action. This would push more often the burden of taking actions on other nodes, thus affecting their battery life. However, this attack is not a relevant threat. The reason is that, if a monitoring node is compromised and it is colluding with the malicious node object of the action, an identical energy-consumption threat would still happen in normal conditions if the colluding node simply never took action (i.e. never let its action timer fire).

Colluding nodes. Kinesis is designed in such a way that it is not possible for compromised nodes to collude with each other. For example, one such malicious node might drop a data packet, and another node might pretend to take a response action just to stop all the other monitors' action timers, but not actually executing such action. This scenario is not possible, since the action timer mechanism expects an action to actually be undertaken in order to have the other timers stopped. In the particular case of proofs of action, the collusion is still not possible, as we discuss next.

Proofs of Action. We investigate the possibility that a compromised node, in the attempt of colluding with an attacker node, might broadcast false proofs of action to stop the other nodes' action times. As we discuss here, the inherent mechanism of proofs of action prevents this scenario from being fruitful for either the attacker or

the colluding node. First, each proof of action must be authenticated, leveraging the cryptographic primitives already in use for the other operations in Kinesis. This will always connect a particular proof of action to the node that generated it. Secondly, while proofs of action are propagated hop-by-hop to all the possibly hidden nodes, they are bound to eventually reach at least one of the action's originator node. Such node would be able to tell that, while the proof of action claims that the originator node performed an action, this never happened. It will thus be able to immediately treat such false proof of action as misbehavior and take action accordingly, by marking the colluding node as malicious and potentially revoking it.

8.10 Discussion

In this section, we analyze the characteristics of Kinesis from various aspects and discuss possible improvements.

False Positives. A false positive occurs if an attack is detected when there is none. In Kinesis, when a monitor observes an anomalous activity by a neighbor node, it does not immediately conclude that this is an attack. It continues to observe the node while taking appropriate response action(s) (conservative or moderate) to mitigate disruption to WSN services. Thus, as long as Kinesis can keep the WSN functional (e.g., send data successfully to the BS) and minimize the disruption, our security goal is achieved.

When the security estimation for the monitored node exceeds a threshold specified in the matched response policy, the monitor may go for an aggressive action, requiring, however, consensus among a minimum number of neighboring monitors. It is highly unlikely that all of these monitors will detect a false attack.

2-hop Knowledge Overhead. From an analytical point of view, handling and storing the additional information about 2-hop knowledge (see Section 8.5.3) results in some memory overhead, strictly dependent on the topology. In the worst-case scenario, where the topology is a fully connected graph of N nodes – and therefore each

node is connected to every other node – then a node will have to store $N-1$ IDs for the neighbors of each one of its $N-1$ direct neighbors, resulting in $(N-1)(N-1) = N^2+1$ values. This theoretical upper bound, however, is relative to the very extreme fully-connected topology described, which is very much unlikely to happen in any real WSN. The lower bound, on the other hand, is that in which a node is either completely isolated (therefore storing 0 IDs of 1-hop and 2-hop neighbors), or only has 1-hop neighbors (e.g. a star topology with a node in the middle and all the others around, therefore storing only $N-1$ IDs for direct neighbors). An average case, more plausible for a common WSN scenario, could be a grid layout for the nodes. In such topology, each node has at most 4 direct neighbors, therefore $4 \cdot (4-1) = 12$ IDs need to be stored per node, which is a very acceptable overhead.

Jamming. An attacker may interrupt Kinesis operation by jamming a part of the network and disabling data communication. We implemented a jamming attack following the method described in [212]. This jamming attack, however, results in no more than 20-30% data loss, which is the same as the data loss in *data_loss* incidents. As part of future work, we will implement stronger jammers able to block the channel completely and will investigate whether Kinesis, in response, can send a top priority message to the BS through the border nodes.

8.11 Related Work

We discuss the work related to Kinesis in following categories: intrusion detection and/or response system for wireless networks, policy specification, daemon selection.

Intrusion Detection and Response System: A number of IDSes have been proposed for wireless and mobile ad-hoc networks (MANET) and WSNs. The majority of these IDSes just raise an alarm or take simple response actions without following any systematic approach. In a pioneering work, Zhang et al. propose a distributed and cooperative IDS for MANET [213]. Each mobile node runs a local IDS agent

that monitors local activities, detects intrusions, and may trigger responses. Neighboring IDS agents cooperate in global intrusion detection when there is inconclusive evidence. The architecture is similar to Kinesis but is more focused on intrusion detection and does not provide a well-designed response framework. Marti et al. propose a mechanism to improve throughput in MANETs in the presence of compromised nodes [29]. They use a watchdog to identify misbehaving nodes and a trust based routing path rating scheme to help routing protocols avoid these nodes. The CONFIDANT protocol aims at detecting and isolating misbehaving nodes, thus making it unattractive to deny cooperation [44]. Trust relationships and routing decision are based on experienced, observed, or reported routing and forwarding behavior of other nodes. The responses in these systems, however, are limited to rerouting data or isolating the misbehaving node.

Ma et al. [45] propose a self adaptive IDS (SAID) for WSN, where three agents, namely monitor, decision, and defense agents, cooperate to defend from intruders in networks. However, the response system in SAID does not follow a systematic approach and the responses are only limited to revoking or suspending a node. Also, the agents need to update a central knowledge base continuously to update the node reputations and to choose response agents accordingly. Hsieh et al. [214] propose an adaptive security design to secure cluster communication via neighbor node authentication, secure link establishment, and send alarms to the BS upon an intrusion. The mechanism proposed by Younis et al. [46] adapts the security provision to the need of the application and the trust of the nodes in the routing path. These mechanisms heavily depend on cryptographic operations and the counterattack is limited to routing path rotation or raising alarms. Taddeo et al. [215] propose a self-adaptation method of security mechanisms. They always start with the highest security level, which may be unnecessary and costly for sensor nodes.

Asim et al. [216] propose an architecture that organizes the WSN nodes in a virtual grid of cells. Each cell has a manager responsible for anomaly detection and recovery. Their approach is not fully distributed and focuses on network failures and

energy related issues, rather than on malicious behaviors or attacks. MALADY is a machine learning-based system that enables nodes to use gathered data to make real-time decisions [217]. However, MALADY aims at the detection and learning process rather than response to attacks. Mamun et al. [218] propose a policy based intrusion detection and response system with a four level hierarchy architecture. Their intrusion response system has a general scope based on customizable policies. However, their only responses are suspend or revocation of the suspect node, and are only applicable to the hierarchical architecture they consider. To the best of our knowledge, Kinesis is the first complete system able to manage automated responses not only to attacks, but also to anomalies with an aim to minimize disruption to WSN services while natural error or an attack progresses.

Policy Specification: A number of policy languages have been proposed for the specification of policies for quality-of-service management within a network [219], privacy management for web users [220], access control in database systems [221], etc. However, these languages serve specific purposes and do not consider the context of WSNs or IRPSes, required to optimally express the response policies. Hence, the resource constrained nature of sensor devices makes it challenging to utilize the typical policy languages used in general purpose networks, database systems, and other domains. We propose a simple and lightweight policy language considering the IRPS specific requirements for WSNs.

Daemon Selection: *Leader* election is a fundamental and well studied problem in fault-tolerant distributed computing. Garcia-Molina [222] first proposed leader election protocols for distributed systems to elect a coordinator node which reorganizes the active nodes after a crash failure and helps them continue the desired tasks. In the context of wired and wireless networks, leader election has a variety of applications, such as key distribution, routing coordination, general control, etc. and a considerable number of leader election protocols [223] has been proposed over the years. In a similar context, many clustering algorithms [224] have been proposed for WSNs to group sensor nodes into clusters and to elect a leader for each cluster for cluster

management and data aggregation. However, these leader election protocols require multiple rounds of group communication and often time synchronization among the participants. In contrast, we propose a *daemon selection* mechanism that selects a node for executing response action in a neighborhood via a self-organized competition among the neighbors. Each node in a neighborhood competes independently using a locally managed *action timer*. We **do not need any time synchronization or message exchanges** among the neighbors.

8.12 Summary

In this chapter, we presented the first incident response and prevention system for WSNs. The system reacts not only to the occurrence of attacks, but also upon anomalous events, so that the WSN remains functional even if the attack progresses. The system is dynamic – as it selects the response actions based on the suspect's security status – and distributed – since it does not require any central authority to trigger the response actions. The simple yet flexible design of the response policies makes the system extensible and thus able to handle new attacks. Kinesis is secure with respect to policy dissemination, storage and execution. The experimental results show that Kinesis achieves high effectiveness in terms of data rate and latency, low redundancy in action executions, and, most importantly, scalability.

9 FUTURE RESEARCH DIRECTIONS

As continuation of the results achieved so far we plan to investigate further directions with the goal of developing security techniques for sensor systems and IoT. Hereafter we present some of these planned research projects.

Prevention through Memory Safety Enforcement on Embedded Devices. As future work, we will extend our implementation of nesCheck to explicitly address temporal safety, and design mechanisms tailored for embedded platforms to enforce it. Moreover, currently, the node is rebooted whenever a dynamic check fails. In the future, we will work on more advanced, programmer-guided recovery mechanisms, with the goal of maintaining the network as functional as possible even in face of memory errors. Lastly, the scalability of the system and further optimization on the overhead are a main goal. We plan to integrate Bounded Model Checking techniques into nesCheck to use advanced formal verification techniques for proving the safety of some seemingly dangerous memory accesses, and therefore further reducing the overhead.

Fine-Grained Diagnosis Techniques for Sensor Systems and IoT. As part of future work, an interesting research issue is to learn and characterize the duration of the interference to discriminate between a naturally occurring interference – such as people walking – and attack-originated interference. Note that our goal for the FGA tool is to provide information useful for response actions to prevent/minimize data losses. Therefore, even when data losses are caused by natural causes, our FGA can provide information useful to potentially reconfigure the network to reduce data losses. For example, if interference due to people walking by is detected, the sensor network could perhaps benefit from additional sensors deployed at different positions.

As further future directions, we plan to extend the FGA tool to cover additional events and anomalies, such as different kinds of jamming. We also plan to investigate the use of learning techniques so that our tool can learn from previous attacks and anticipate future ones. Finally, automated intelligent tools might help the network administrator in tweaking the various parameters offered by the FGA tool to tailor it to the specific features of the WSN of interest.

Concerning the statistical model for FGA, we plan on further evaluating the effectiveness of our approach on real testbeds, as well as comparing the system performance with the performance of tools based using machine learning methods (e.g. neural networks or Naive Bayes classifiers). Moreover, we will investigate the potential of using different P_{FA} values for different links based on various criteria, such as distance or expected QoS.

Mobility-aware Fine-Grained Analysis. The diagnosis techniques presented in this dissertation with the FGA tools are effective in differentiating node- and link-related incidents and, in case of interference, accurately locate the source of noise. However, our technique assumes a static network of sensor nodes. While this is true for many WSNs, several applications of sensor systems leverage mobile nodes whose position changes over time. This includes, for example, critical scenarios for military operation, drones, MANETs, and VANETs. For this reason, one of our research work directions includes enhancing the design of our FGA approach to make it applicable to mobile sensor systems. Our initial design for a mobility-aware FGA approach leverages a 2-hop knowledge of a nodes neighbors in order to construct a set of geometric constraints that can help in localizing the direct neighbors with respect to a fixed system of coordinates. However, the non-linearity of some of the parameters used in the design of the FGA such as the signal strength indicator makes this a hard problem. We will continue working in this direction, as well as investigating other potential alternative design to tackle this important problem.

10 CONCLUSIONS

Our research work aims at securing sensor systems and IoT through the development of new security techniques, as well as the adaptation and enhancement of existing ones.

We argue that the intrinsic characteristics of the sensor and IoT domain expand the attack surface of computer and communication systems. Existing security techniques need to be analyzed, extended and modified in order to efficiently and effectively achieve security across heterogeneous and constrained scenarios, throughout all the four phases of hardening, monitoring, diagnosing, and recovering. A thoughtful leveraging of such domain characteristics enables an effective use of techniques that are less suitable for traditional networks and systems, such as overhearing-based monitoring, whitelist-based anomaly detection, or whole program analysis. The solutions and results we presented in this dissertation substantiate our claim, and compose an overall security framework able to ensure security for sensor systems and IoT.

Overall, we identified and addressed four security phases – namely "Prepare and Prevent", "Monitor and Detect", "Diagnose and Understand", "React, Recover and Fix" – with various security solutions.

The initial hardening steps before the deployment of constrained devices must address the security of the firmware in face of memory vulnerabilities. We designed nesCheck, a novel whole program analysis-based approach that combines static analysis and dynamic checking to efficiently enforce memory safety on existing embedded software, without requiring any source modification.

Since protections such as that offered by nesCheck come at a cost, determining the optimal allocation plan for security measures becomes very important for real-time, constrained systems. Therefore, we presented OptAll, a game-theory-based method to compute the optimal security resource allocation plan through a Pareto

optimization problem, taking into account the available security resources and their capabilities, their fixed cost, and runtime energy consumption, how critical different areas of the network are, as well as the risk associated with successful attacks on them.

The operation of IoT devices and sensors must be continuously monitored to detect anomalies and attacks, but the heterogeneity of systems and protocols make it a challenging task. On the other hand, knowledge about the features of the network and entities, as well as about the security measures in place (provided by our solutions in the Prepare and Prevent phase), can be leveraged to achieve a more accurate detection. We thus first developed a taxonomy to better represent the relationships between IoT network features and related attacks, and then designed and developed Kalis, an overhearing-based, self-adapting, knowledge-driven IDS for IoT able to detect attacks in real time across heterogeneous IoT systems.

While Kalis detects attacks to the IoT network, the exposure to the untrusted Internet makes the devices desirable for botnets; accounting for the typical communication patterns of IoT, we designed a centralized router-based defense, Heimdall, that prevents botnet DDoS attacks through a whitelist-based anomaly detection technique tailored to IoT devices.

For the monitoring solutions we developed, it is critical to perform an accurate diagnosis of the detected security incidents in order to provide effective response actions. For the important class of attacks of packet losses in sensor networks, we developed a Fine-Grained Analysis (FGA) tool that leverages resident packet parameters to determine the incident's root cause and, in case of interference, locates the source of jamming.

As the accuracy of the FGA tool relies on the correct choice of some system parameters, we extended this work with a statistical-based approach for determining optimal system thresholds by exploiting the variances of RSSI and LQI.

The accurate diagnosis information provided by our previously introduced solutions make it possible to take automatic, effective response actions to security inci-

dents in order to maintain the network functional. We therefore designed Kinesis, a security incident response system for WSNs aimed at keeping the network functional despite anomalies or attacks and to recover from attacks without significant interruption. Its overhearing-based distributed strategy makes the system efficient and scalable, achieving load-balancing and redundant action optimization, while maintaining a fully-distributed design.

As part of our research effort, all the developed solutions fit in an overall security framework to employ them in concert and provide a holistic approach at securing sensor systems and the IoT.

REFERENCES

REFERENCES

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[2] James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. Disruptive technologies: Advances that will transform life, business, and the global economy. `http://www.mckinsey.com/insights/business_technology/disruptive_technologies`, May 2013.

[3] Rob van der Meulen. Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. `http://www.gartner.com/newsroom/id/3165317`, November 2015.

[4] Yong Ho Hwang. IoT security & privacy: Threats and challenges. In *Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security*, IoTPTS '15, page 1, New York, NY, USA, 2015. ACM.

[5] K. Zhao and L. Ge. A survey on the internet of things security. In *Proceedings of the 9th International Conference on Computational Intelligence and Security (CIS)*, pages 663–667, December 2013.

[6] Zhi-Kai Zhang, Michael Cheng Yi Cho, Chia-Wei Wang, Chia-Wei Hsu, Chong-Kuan Chen, and Shiuhpyng Shieh. IoT security: ongoing challenges and research opportunities. In *IEEE 7th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 230–234. IEEE, 2014.

[7] M. U. Farooq, Muhammad Waseem, Anjum Khairi, and Sadia Mazhar. A critical analysis on the security concerns of internet of things. *International Journal of Computer Applications*, 111(7), 2015.

[8] JeongGil Ko, Chenyang Lu, M.B. Srivastava, J.A. Stankovic, A. Terzis, and M. Welsh. Wireless sensor networks for healthcare. *Proceedings of the IEEE*, 98(11):1947–1960, 2010.

[9] Alex Wright. Hacking cars. *Communications of the ACM*, 54(11):18–19, November 2011.

[10] Sudhir K Bansal. Linux worm targets internet-enabled home appliances to mine cryptocurrencies. `http://thehackernews.com/2014/03/linux-worm-targets-internet-enabled.html`, 2014. Accessed: May 2016.

[11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, page 6, Berkeley, CA, USA, 2011. USENIX Association.

[12] Antone Gonsalves. New toolkit seeks routers, internet of things for DDoS botnet. `http://www.csoonline.com/article/2687653/data-protection/new-toolkit-seeks-/routers-internet-of-things-for-ddos-botnet.html`, 2014. Accessed: May 2016.

[13] William Alexander. Barnaby Jack could hack your pacemaker and make your heart explode. `http://www.vice.com/en\_ca/read/i-worked-out-how-to-remotely-weaponise-a-pacemaker`, June 2013.

[14] Daniel Miessler. HP study reveals 70 percent of internet of things devices vulnerable to attack. `http://h30499.www3.hp.com/t5/Fortify-Application-Security/HP-Study-Reveals-70-Percent-of-Internet-of-Things-Devices/ba-p/6556284#.VH4faTHF9Zg`, July 2014.

[15] Daniele Midi, Mathias Payer, and Elisa Bertino. Static analysis and dynamic instrumentation for nesC memory safety. *under submission*, 2017.

[16] George C Necula, Scott McPeak, and Westley Weimer. CCured: Type-safe retrofitting of legacy code. *ACM SIGPLAN Notices*, 37(1):128–139, 2002.

[17] Santosh Nagarakatte, Jianzhou Zhao, Milo MK Martin, and Steve Zdancewic. SoftBound: highly compatible and complete spatial memory safety for C. In *ACM Sigplan Notices*, volume 44, pages 245–258. ACM, 2009.

[18] Nathan Cooprider, Will Archer, Eric Eide, David Gay, and John Regehr. Efficient memory safety for TinyOS. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, pages 205–218. ACM, 2007.

[19] Antonino Rullo, Daniele Midi, Edoardo Serra, and Elisa Bertino. Strategic security resource allocation for internet of things. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016.

[20] Edoardo Serra, Sushil Jajodia, Andrea Pugliese, Antonino Rullo, and VS Subrahmanian. Pareto-optimal adversarial defense of enterprise systems. *ACM Transactions on Information and System Security (TISSEC)*, 2015.

[21] Rinku Dewri, Indrajit Ray, Nayot Poolsappasit, and Darrell Whitley. Optimal security hardening on attack tree models of networks: a cost-benefit analysis. *International Journal of Information Security*, 2012.

[22] Eitan Altman, Konstantin Avrachenkov, and Andrey Gamaev. Jamming in wireless networks: The case of several jammers. In *Proceedings of the 1st ICST International Conference on Game Theory for Networks*, 2009.

[23] Quanyan Zhu, Husheng Li, Zhu Han, and Tamer Basar. A stochastic game model for jamming in multi-channel cognitive radio systems. In *IEEE ICC*, 2010.

[24] Zhu Han, Ninoslav Marina, Mérouane Debbah, and Are Hjørungnes. Physical layer security game: How to date a girl with her boyfriend on the same table. In *Proceedings of the 1st ICST International Conference on Game Theory for Networks*, 2009.

[25] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computation*, 2012.

[26] Shahid Raza, Linus Wallgren, and Thiemo Voigt. SVELTE: Real-time intrusion detection in the internet of things. *Ad Hoc Networks*, 11(8):2661–2674, November 2013.

[27] Ioannis Krontiris, Thanassis Giannetsos, and Tassos Dimitriou. LIDeA: A distributed lightweight intrusion detection architecture for sensor networks. In *International Conference on Security and Privacy in Communication Networks (SecureComm)*, pages 20:1–20:10, 2008.

[28] Y. Ponomarchuk and Dae-Wha Seo. Intrusion detection based on traffic analysis in wireless sensor networks. In *Annual Wireless and Optical Communications Conference*, pages 1–7, 2010.

[29] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *International Conference on Mobile Computing and Networking (MobiCom)*, pages 255–265, 2000.

[30] Daniele Midi, Antonino Rullo, Anand Mudgerikar, and Elisa Bertino. Kalis: A system for knowledge-driven adaptable intrusion detection for the internet of things. *under submission*, 2017.

[31] Daniele Midi, Anand Mudgerikar, Javid Habibi, and Elisa Bertino. Heimdall: Mitigating the internet of insecure things. *under submission*, 2017.

[32] I. Murynets and R.P. Jover. Anomaly detection in cellular machine-to-machine communications. In *IEEE International Conference on Communications (ICC)*, pages 2138–2143, June 2013.

[33] C. M. Liu, S. Y. Chen, Y. Zhang, R. Chen, and K. L. Guo. An IoT anomaly detection model based on artificial immunity. *Advanced Materials Research*, January 2012.

[34] Fabio Gonzalez. A study of artificial immune systems applied to anomaly detection. *University of Memphis Dissertations*, 2003.

[35] Daniele Midi and Elisa Bertino. Node or link? fine-grained analysis of packet-loss attacks in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 12(2):8, 2016.

[36] Lili Qiu, Paramvir Bahl, Ananth Rao, and Lidong Zhou. Troubleshooting wireless mesh networks. *ACM SIGCOMM Computer Communication Review*, 36(5):17–28, 2006.

[37] Krishna N. Ramach, Elizabeth M. Belding-royer, and Kevin C. Almeroth. Damon: A distributed architecture for monitoring multi-hop mobile networks. In *Proceedings of IEEE SECON*, 2004.

[38] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing, 2003.

[39] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *ACM MobiCom*, pages 114–128. ACM Press, 2004.

[40] Sookhyun Yang, Sudarshan Vasudevan, and Jim Kurose. Witness based witness-based detection of forwarding misbehaviors in wireless networks. In *UMass Computer Science Technical Report UM-CS-2009-001*, 2009.

[41] Jianxia Ning, Shailendra Singh, Konstantinos Pelechrinis, Bin Liu, Srikanth V. Krishnamurthy, and Ramesh Govindan. Forensic analysis of packet losses in wireless networks. In *ICNP*, pages 1–10, 2012.

[42] Daniele Midi, Antonio Tedeschi, Francesco Benedetto, and Elisa Bertino. Statistically-enhanced fine-grained diagnosis of packet losses. In *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 748–753. IEEE, 2015.

[43] Daniele Midi, Salmin Sultana, and Elisa Bertino. A system for response and prevention of security incidents in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2017.

[44] Sonja Buchegger and Jean-Yves Boudec. Performance analysis of the CONFIDANT protocol. In *ACM International Symposium on Mobile Ad Hoc Networking (MobiHoc)*, pages 226–236, 2002.

[45] Jianqing Ma, Shiyong Zhang, Yiping Zhong, and Xiaowen Tong. SAID: A self-adaptive intrusion detection system in wireless sensor networks. In *International Conference on Information Security Applications*, pages 60–73, 2007.

[46] M. Younis, N. Krajewski, and O. Farrag. Adaptive security provision for increased energy efficiency in wireless sensor networks. In *IEEE Conference on Local Computer Networks*, pages 999–1005, 2009.

[47] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An operating system for sensor networks. In Werner Weber, JanM. Rabaey, and Emile Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005.

[48] Philip Levis. Experiences from a decade of TinyOS development. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 207–220, Berkeley, CA, USA, 2012. USENIX Association.

[49] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 1–11, New York, NY, USA, 2003. ACM.

[50] Aurélien Francillon and Claude Castelluccia. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pages 15–26, New York, NY, USA, 2008. ACM.

[51] Thanassis Giannetsos, Tassos Dimitriou, Ioannis Krontiris, and Neeli R. Prasad. Arbitrary code injection through self-propagating worms in Von Neumann architecture devices. *Computation Journal*, 53(10):1576–1593, December 2010.

[52] Aurélien Francillon, Daniele Perito, and Claude Castelluccia. Defending embedded systems against control flow attacks. In *Proceedings of the 1st ACM Workshop on Secure Execution of Untrusted Code*, pages 19–26. ACM, 2009.

[53] Thanassis Giannetsos and Tassos Dimitriou. Spy-sense: Spyware tool for executing stealthy exploits against sensor networks. In *Proceedings of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, HotWiSec '13, pages 7–12, New York, NY, USA, 2013. ACM.

[54] Travis Goodspeed. Stack overflow exploits for wireless sensor networks over 802.15.4. 2008.

[55] Bo Sun, D. Shrestha, Guanhua Yan, and Yang Xiao. Self-propagate mal-packets in wireless sensor networks: Dynamics and defense implications. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5, November 2008.

[56] Yi Yang, Sencun Zhu, and Guohong Cao. Improving sensor network immunity under worm attacks: a software diversity approach. In *Proceedings of the 9th ACM International symposium on Mobile Ad Hoc Networking and Computing*, pages 149–158. ACM, 2008.

[57] Memsic. TelosB datasheet. `http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf`.

[58] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. CETS: Compiler enforced temporal safety for C. *SIGPLAN Notices*, 45(8):31–40, August 2010.

[59] Robin Züger. Paging in TinyOS, 2006.

[60] Ron Cytron, Jeanne Ferrante, Barry K Rosen, Mark N Wegman, and F Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(4):451–490, 1991.

[61] Trevor Jim, J. Greg Morrisett, Dan Grossman, Michael W. Hicks, James Cheney, and Yanling Wang. Cyclone: A safe dialect of C. In *Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference*, ATEC '02, pages 275–288, Berkeley, CA, USA, 2002. USENIX Association.

[62] Todd M. Austin, Scott E. Breach, and Gurindar S. Sohi. Efficient detection of all pointer and array access errors. In *Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation*, PLDI '94, pages 290–301, New York, NY, USA, 1994. ACM.

[63] Joe Devietti, Colin Blundell, Milo M. K. Martin, and Steve Zdancewic. Hardbound: Architectural support for spatial safety of the C programming language. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 103–114, New York, NY, USA, 2008. ACM.

[64] Harish Patil and Charles Fischer. Low-cost, concurrent checking of pointer and array accesses in C programs. *Software Practice and Expertise*, 27(1):87–110, January 1997.

[65] Wei Xu, Daniel C. DuVarney, and R. Sekar. An efficient and backwards-compatible transformation to ensure memory safety of C programs. In *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, SIGSOFT '04/FSE-12, pages 117–126, New York, NY, USA, 2004. ACM.

[66] Clang. Clang: A C language family frontend for LLVM. `http://clang.llvm.org/`.

[67] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, March 2004.

[68] Doina Bucur and Marta Z Kwiatkowska. Software verification for TinyOS. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 400–401. ACM, 2010.

[69] Doina Bucur. Intelligible TinyOS sensor systems: Explanations for embedded software. In *Modeling and Using Context*, pages 54–66. Springer, 2011.

[70] Raimondas Sasnauskas, Jó Ágila Bitsch Link, Muhammad Hamad Alizai, and Klaus Wehrle. Kleenet: automatic bug hunting in sensor network applications. In *Proceedings of the 6th ACM Conference on Embedded network sensor systems*, pages 425–426. ACM, 2008.

[71] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st International Conference on Embedded networked sensor systems*, pages 126–137. ACM, 2003.

[72] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pages 188–200, New York, NY, USA, 2004. ACM.

[73] Vinaitheerthan Sundaram, Patrick Eugster, and Xiangyu Zhang. Prius: Generic hybrid trace compression for wireless sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 183–196, 2012.

[74] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 95–107, 2004.

[75] Memsic. MicaZ datasheet. `http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf`.

[76] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. *Symbolic model checking without BDDs*. Springer, 1999.

[77] SoftBound website. `http://www.cis.upenn.edu/acg/softbound/`.

[78] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. SoK: Eternal war in memory. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 48–62, Washington, DC, USA, 2013. IEEE Computer Society.

[79] Jeremy Condit, Matthew Harren, Zachary Anderson, David Gay, and George C Necula. Dependent types for low-level programming. In *Programming Languages and Systems*, pages 520–535. Springer, 2007.

[80] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176. Springer, 2004.

[81] Heinrich von Stackelberg, Damien Bazin, Rowland Hill, and Lynn Urch. *Market Structure and Equilibrium*. Springer, 2010.

[82] Rodrigo Roman, Cristina Alcaraz, Javier Lopez, and Nicolas Sklavos. Key management systems for sensor networks in the context of the internet of things. *Computers & Electrical Engineering*, 37(2):147–159, 2011.

[83] Lal C Godara. Application of antenna arrays to mobile communications: Beam-forming and direction-of-arrival considerations. *Proceedings of the IEEE*, 85(8):1195–1245, 1997.

[84] Asis Nasipuri and Kai Li. A directionality based location discovery scheme for wireless sensor networks. In *Proceedings of 1st ACM International Workshop on Wireless sensor networks and applications*. ACM, 2002.

[85] Chanatip Tumrongwittayapak and Ruttikorn Varakulsiripunth. Detecting sinkhole attack and selective forwarding attack in wireless sensor networks. In *Information, Communications and Signal Processing*, 2009.

[86] Ferdinand Brasser, Brahim El Mahjoub, Ahmad Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. TyTAN: Tiny trust anchor for tiny devices. In *DAC*, pages 1–6. IEEE, 2015.

[87] Srdjan Čapkun, Levente Buttyán, and Jean Pierre Hubaux. Sector: secure tracking of node encounters in multi-hop wireless networks. In *Proceedings of 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2003.

[88] Yih Chun Hu, Adrian Perrig, and David B Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *INFOCOM 2003*. IEEE, 2003.

[89] Kalpana Sharma and MK Ghose. Wireless sensor networks: An overview on its security threats. *IJCA, Special Issue on Mobile Ad-hoc Networks MANETs*, pages 42–45, 2010.

[90] Issa Khalil, Saurabh Bagchi, and Ness B Shroff. LITEWORP: a lightweight countermeasure for the wormhole attack in multihop wireless networks. In *Dependable Systems and Networks*, pages 612–621. IEEE, 2005.

[91] Dazhi Zhang and Donggang Liu. DataGuard: Dynamic data attestation in wireless sensor networks. In *DSN, 2010*. IEEE, 2010.

[92] Laurent Eschenauer and Virgil D Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 41–47. ACM, 2002.

[93] Achille Messac, Amir Ismail-Yahaya, and Christopher A Mattson. The normalized normal constraint method for generating the Pareto frontier. *Structural and multidisciplinary optimization*, 25(2):86–98, 2003.

[94] Imad Jawhar, Nader Mohamed, and Liren Zhang. A distributed topology discovery algorithm for linear sensor networks. In *1st IEEE International Conference on Communications in China (ICCC)*, pages 775–780. IEEE, 2012.

[95] Alireza A Nezhad, Dimitris Makrakis, and Ali Miri. Anonymous topology discovery for multihop wireless sensor networks. In *Proceedings of the 3rd ACM Workshop on QoS and Security for Wireless and Mobile Networks*, pages 78–85. ACM, 2007.

[96] IBM ILOG. CPLEX 12.5, 2011.

[97] Quanyan Zhu, Linda Bushnell, and Tamer Basar. Game-theoretic analysis of node capture and cloning attack with multiple attackers in wireless sensor networks. In *CDC*, pages 3404–3411. IEEE, 2012.

[98] Ho Ting Cheng and Weihua Zhuang. Pareto optimal resource management for wireless mesh networks with qos assurance: joint node clustering and subcarrier allocation. *IEEE Transactions on Wireless Communications*, 2009.

[99] Liang Zhou and Han Chieh Chao. Multimedia traffic security architecture for the internet of things. *IEEE Networking*, 25(3):35–40, 2011.

[100] Shahid Raza, Simon Duquennoy, Joel Höglund, Utz Roedig, and Thiemo Voigt. Secure communication for the internet of things: a comparison of link-layer security and ipsec for 6lowpan. *Security and Communication Networks*, 2012.

[101] Thang N Dinh, Ying Xuan, My T Thai, EK Park, and Taieb Znati. On approximation of new optimization methods for assessing network vulnerability. In *Proceedings of IEEE INFOCOM*, 2010.

[102] Peter V Marsden. Egocentric and sociocentric measures of network centrality. *Social Networks*, 24(4):407–422, 2002.

[103] Anne Marie Kermarrec, Erwan Le Merrer, Bruno Sericola, and Gilles Trédan. Second order centrality: Distributed assessment of nodes criticity in complex networks. *Computer Communications*, 2011.

[104] Ashwin Arulselvan, Clayton W Commander, Lily Elefteriadou, and Panos M Pardalos. Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 36(7), 2009.

[105] Devesh Jinwala, Dhiren Patel, and Kankar Dasgupta. FlexiSec: a configurable link layer security architecture for wireless sensor networks. *Journal of Information Assurance and Security*, 2012.

[106] Shahid Raza, Simon Duquennoy, Joel Hglund, Utz Roedig, and Thiemo Voigt. Secure communication for the internet of things: a comparison of link-layer security and IPsec for 6LoWPAN. *Security and Communication Networks*, 7(12):2654–2668, 2014.

[107] Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Brünig, and Georg Carle. DTLS based security and two-way authentication for the internet of things. *Ad Hoc Networks*, 11(8):2710–2723, 2013.

[108] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6LoWPAN with compressed IPsec. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–8, June 2011.

[109] Shahid Raza, Hossein Shafagh, Kasun Hewage, René Hummen, and Thiemo Voigt. Lithe: Lightweight secure CoAP for the internet of things. *IEEE Sensors Journal*, 13(10):3711–3720, 2013.

[110] Thomas Kothmayr, Wen Hu, Corinna Schmitt, Michael Bruenig, and Georg Carle. Poster: Securing the internet of things with DTLS. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 345–346. ACM, 2011.

[111] Alan Grau. Intrusion detection software lowers internet of things risk. `http://www.controleng.com/single-article/intrusion-detection-software-lowers-internet-of-things-iot-risk/e1ca58c3af94e41f26bc4836c21803f5.html`, 2015. Accessed: May 2016.

[112] Cai Ming Liu, Run Chen, and Chao Chen. An artificial immune-based distributed intrusion detection model for the internet of things. In *Advanced Research on Material Engineering, Architectural Engineering and Informatization*, volume 366 of *Advanced Materials Research*, pages 165–168. Trans Tech Publications, January 2012.

[113] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 1, pages 229–238, 1999.

[114] The Security Ledger. IDS and the IoT: Snort creator marty roesch on securing the internet of things. `https://securityledger.com/2014/04/ids-and-the-iot-snort-creator-marty-roesch-on-securing-the-internet-of-things/`.

[115] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[116] IEEE. IEEE 802.15 WPAN Task Group 4 (TG4). `http://www.ieee802.org/15/pub/TG4.html`.

[117] ZigBee Alliance and others. Zigbee specification, 2006.

[118] Jonathan Hui, David Culler, and Samita Chakrabarti. 6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture. *IPSO Alliance White Paper*, 3, 2009.

[119] A. Mishra, K. Nadkarni, and A. Patcha. Intrusion detection in wireless ad hoc networks. *IEEE Wireless Communications*, 11(1):48–60, February 2004.

[120] Yi-an Huang and Wenke Lee. A cooperative intrusion detection system for ad hoc networks. In *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN '03, pages 135–147, New York, NY, USA, 2003. ACM.

[121] David Miller, Shon Harris, Allen Harper, Stephen VanDyke, and Chris Blask. *Security information and event management (SIEM) implementation*. McGraw Hill Professional, 2010.

[122] Farzad Sabahi and Ali Movaghar. Intrusion detection: A survey. In *Proceedings of the 3rd International Conference on Systems and Networks Communications*, pages 23–26. IEEE, 2008.

[123] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1):18–28, 2009.

[124] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ACM SenSys, pages 1–14, 2009.

[125] Loh Chin Choong Desmond, Cho Chia Yuan, Tan Chung Pheng, and Ri Seng Lee. Identifying unique devices through wireless fingerprinting. In *Proceedings of the 1st ACM Conference on Wireless Network Security*, WiSec '08, pages 46–55, New York, NY, USA, 2008. ACM.

[126] J. Wang, G. Yang, Y. Sun, and S. Chen. Sybil attack detection based on RSSI for wireless sensor network. In *International Conference on Wireless Communications, Networking and Mobile Computing*, pages 2684–2687, September 2007.

[127] V Manjula and Dr C Chellappan. Replication attack mitigations for static and mobile WSN. *arXiv preprint arXiv:1103.3378*, 2011.

[128] Vern Paxson, Scott Campbell, Jason Lee, et al. Bro intrusion detection system. Technical report, Lawrence Berkeley National Laboratory, 2006.

[129] Rung-Ching Chen, Chia-Fen Hsieh, and Yung-Fa Huang. A new method for intrusion detection on hierarchical wireless sensor networks. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '09, pages 238–245, New York, NY, USA, 2009. ACM.

[130] M. Tiwari, K. V. Arya, R. Choudhari, and K. S. Choudhary. Designing intrusion detection to detect black hole and selective forwarding attack in WSN based on local information. In *Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology*, pages 824–828, November 2009.

[131] K. Gerrigagoitia, R. Uribeetxeberria, U. Zurutuza, and I. Arenaza. Reputation-based intrusion detection system for wireless sensor networks. In *Complexity in Engineering (COMPENG), 2012*, pages 1–5, June 2012.

[132] B. Sun, L. Osborne, Y. Xiao, and S. Guizani. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wireless Communications*, 14(5):56–63, October 2007.

[133] Nabil Ali Alrajeh, Shafiullah Khan, and Bilal Shams. Intrusion detection systems in wireless sensor networks: a review. *International Journal of Distributed Sensor Networks*, 2013, 2013.

[134] T Winter, P Thubert, A Brandt, J Hui, R Kelsey, P Levis, K Pister, R Struik, JP Vasseur, and R Alexander. RPL: IPv6 routing protocol for low-power and lossy networks. *RFC*, 2012.

[135] C. Liu, J. Yang, R. Chen, Y. Zhang, and J. Zeng. Research on immunity-based intrusion detection technology for the internet of things. In *International Conference on Natural Computation (ICNC)*, volume 1, pages 212–216, July 2011.

[136] C. Jun and C. Chi. Design of complex event-processing IDS in internet of things. In *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, pages 226–229, January 2014.

[137] Sachin Babar, Parikshit Mahalle, Antonietta Stango, Neeli Prasad, and Ramjee Prasad. *Proposed Security Model and Threat Taxonomy for the Internet of Things*, pages 420–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[138] Anthéa Mayzaud, Rémi Badonnel, and Isabelle Chrisment. A taxonomy of attacks in RPL-based internet of things. *International Journal of Network Security*, 2016.

[139] Antone Gonsalves. New toolkit seeks routers, internet of things for DDoS botnet. `http://www.csoonline.com/article/2687653/data-protection/new-toolkit-seeks-/routers-internet-of-things-for-ddos-botnet.html`, September 2014.

[140] Ping Wang, S. Sparks, and C.C. Zou. An advanced hybrid peer-to-peer botnet. *IEEE Transactions on Dependable and Secure Computing*, 7(2):113–127, April 2010.

[141] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, pages 189–202. ACM, 2006.

[142] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. A self-repairing peer-to-peer system resilient to dynamic adversarial churn. In *Peer-to-Peer Systems IV*, pages 13–23. Springer, 2005.

[143] Linksys. Linksys wrt1900ac ac1900 dual-band smart WiFi wireless router. `http://www.linksys.com/us/p/P-WRT1900AC/`.

[144] OpenWRT. `https://openwrt.org/`.

[145] OpenWRT Community. Opkg package manager. `http://wiki.openwrt.org/doc/techref/opkg`.

[146] Asus. AsusWRT. `http://www.asus.com/ASUSWRT/`.

[147] VirusTotal Community. VirusTotal. `www.virustotal.com`.

[148] VirusTotal. Credits & acknowledgements. `https://www.virustotal.com/en/about/credits/`.

[149] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, pages 9:1–9:9, Berkeley, CA, USA, 2008. USENIX Association.

[150] Janessa Rivera. Gartner says 4.9 billion connected "things" will be in use in 2015. http://www.gartner.com/newsroom/id/2905717, November 2014.

[151] Carlos Gañán, Orcun Cetin, and Michel van Eeten. An empirical analysis of zeus c&c lifetime. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 97–108, New York, NY, USA, 2015. ACM.

[152] Daniel Miessler. Securing the internet of things: Mapping attack surface areas using the OWASP IoT top 10. https://drive.google.com/file/d/0B52IUvO0LP6OdW1HMjRpM3VVUVE/view.

[153] Hewlett-Packard. How safe are home security systems? http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-7342ENW.pdf, February 2015.

[154] David Dagon, Cliff Changchun Zou, and Wenke Lee. Modeling botnet propagation using time zones. In *NDSS*, volume 6, pages 2–13, 2006.

[155] Netfilter Organization. The netfilter.org "iptables" project. http://www.netfilter.org/projects/iptables/index.html.

[156] Nest Labs. Nest. https://nest.com/.

[157] August. August smart lock. http://august.com/.

[158] August. August smart connect. http://support.august.com/customer/portal/articles/1878915-august-connect.

[159] Inc Amazon.com. Amazon dash button. https://www.amazon.com/b/?node=10667898011&sort=date-desc-rank&lo=digital-text.

[160] Netgear. Arlo. https://www.arlo.com/en-us/.

[161] Inc LiFi Labs. Lifx. https://www.lifx.com/.

[162] Hyenae. Network packet generator tool. https://sourceforge.net/projects/hyenae/. Accessed: Jan 2016.

[163] Iperf Community. What is iperf/iperf3. https://iperf.fr/.

[164] Hardkernal Company. Odroid-c1. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G141578608433&tab_idx=1.

[165] Hardkernal Company. Odroid-xu3. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127&tab_idx=1.

[166] C. Onwubiko. Functional requirements of situational awareness in computer network security. In *IEEE International Conference on Intelligence and Security Informatics*, pages 209–213, June 2009.

[167] Cyril Onwubiko and Thomas Owens. *Situational Awareness in Computer Network Defense: Principles, Methods and Applications*. IGI Global, 2012.

[168] Hyo-Sang Lim, G. Ghinita, E. Bertino, and M. Kantarcioglu. A game-theoretic approach for high-assurance of data trustworthiness in sensor networks. In *Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE)*, pages 1192–1203, April 2012.

[169] A.S.K. Pathan, Hyung-Woo Lee, and Choong Seon Hong. Security in wireless sensor networks: issues and challenges. In *The 8th International Conference on Advanced Communication Technology (ICACT)*, volume 2, February 2006.

[170] Yong Wang, Garhan Attebury, and Byrav Ramamurthy. A survey of security issues in wireless sensor networks. *IEEE Communications Surveys Tutorials*, 2006.

[171] Tanveer Zia and Albert Zomaya. Security issues in wireless sensor networks. In *Systems and Networks Communications, 2006. ICSNC'06. International Conference on*, pages 40–40. IEEE, 2006.

[172] Mohammad Maifi Hasan Khan, Hieu K. Le, Michael LeMay, Parya Moinzadeh, Lili Wang, Yong Yang, Dong K. Noh, Tarek Abdelzaher, Carl A. Gunter, Jiawei Han, and Xin Jin. Diagnostic powertracing for sensor node failure analysis. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, pages 117–128, New York, NY, USA, 2010. ACM.

[173] F. Mouton and H.S. Venter. A prototype for achieving digital forensic readiness on wireless sensor networks. In *AFRICON, 2011*, pages 1–6, September 2011.

[174] Ioannis Krontiris, Tassos Dimitriou, and Felix C. Freiling. Towards intrusion detection in wireless sensor networks. In *Proceedings of the 13th European Wireless Conference*, 2007.

[175] Chih fan Hsin. A distributed monitoring mechanism for wireless sensor networks. In *ACM Workshop on Wireless Security*, pages 57–66. Spring, 2002.

[176] R. Roman, Jianying Zhou, and J. Lopez. Applying intrusion detection systems to wireless sensor networks. In *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference*, volume 1, pages 640–644, January 2006.

[177] Salmin S, Gabriel G, E Bertino, and M Shehab. A lightweight secure provenance scheme for wireless sensor networks. *Intl. Conference on Parallel and Distributed Systems*, 2012.

[178] Qingjiang Shi, Chen He, Hongyang Chen, and Lingge Jiang. Distributed wireless sensor network localization via sequential greedy optimization algorithm. *IEEE Transactions on Signal Processing*, 58(6):3328–3340, June 2010.

[179] Tom M Apostol and Mamikon A Mnatsakanian. Centroids constructed graphically. *Mathematics Magazine*, pages 201–210, 2004.

[180] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 364–369, April 2005.

[181] Carlo Alberto Boano, Zhitao He, Yafei Li, Thiemo Voigt, Marco Zuniga, and Andreas Willig. Controllable radio interference for experimental and testing purposes in wireless sensor networks. In *Proceedings of the 4th IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*, 2009.

[182] G. V. Zàruba, M. Huber, F. A. Kamangar, and I. Chlamtac. Indoor location tracking using RSSI readings from a single WiFi access point. *Wireless Networks*, 13(2):221–235, April 2007.

[183] A. Thottam Parameswaran, I. Husain M, and S. Upadhyaya. Is RSSI a reliable parameter in sensor localization algorithms? An experimental study. *28th IEEE SRDS F2DA Workshop*, Sep 2009.

[184] Giovanni Zanca, Francesco Zorzi, Andrea Zanella, and Michele Zorzi. Experimental comparison of RSSI-based localization algorithms for indoor wireless sensor networks. In *Proceedings of the Workshop on Real-world Wireless Sensor Networks*, REALWSN '08, pages 1–5, New York, NY, USA, 2008. ACM.

[185] Kannan Srinivasan and Philip Levis. RSSI is under appreciated. In *Proceedings of the 3rd Workshop on Embedded Networked Sensors (EmNets)*, 2006.

[186] Maurizio Bocca, Ossi Kaltiokallio, Neal Patwari, and Suresh Venkatasubramanian. Multiple target tracking with rf sensor networks. *IEEE Transactions on Mobile Computing*, 13(8):1787–1800, 2014.

[187] Yeong-Sheng Chen, Tai-Lin Chin, and Yi-Chen Huang. Collaborative localization in wireless sensor networks based on dependable RSSI. In *15th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 341–347, September 2012.

[188] Ugur Bekcibasi and Mahmut Tenruh. Increasing RSSI localization accuracy with distance reference anchor in wireless sensor networks. *Acta Polytechnica Hungarica*, 11(8), 2014.

[189] Bilal Shebaro, Daniele Midi, and Elisa Bertino. Fine-grained analysis of packet loss symptoms in wireless sensor networks. In *Proceedings of IEEE SECON*, 2014.

[190] Yin Chen and Andreas Terzis. On the mechanisms and effects of calibrating RSSI measurements for 802.15.4 radios. In *Proceedings of the 7th European Conference on Wireless Sensor Networks*, EWSN'10, pages 256–271, Berlin, Heidelberg, 2010. Springer-Verlag.

[191] E. Moulines and K. Choukri. Time-domain procedures for testing that a stationary time-series is gaussian. *IEEE Transactions on Signal Processing*, 44(8):2010–2025, August 1996.

[192] E. Guzzon, F. Benedetto, and G. Giunta. A new test for initial code acquisition of correlated cells. *IEEE Transactions on Vehicular Technology*, 62(5):2349–2358, June 2013.

[193] F. Benedetto, G. Giunta, E. Guzzon, and M. Renfors. Effective monitoring of freeloading user in the presence of active user in cognitive radio networks. *IEEE Transactions on Vehicular Technology*, 63(5):2443–2450, June 2014.

[194] Umeshwar Dayal. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Publishers Inc., 1994.

[195] Sangwon Hyun, Peng Ning, An Liu, and Wenliang Du. Seluge: Secure and DoS-resistant code dissemination in wireless sensor networks. In *International Conference on Information Processing in Sensor Networks (IPSN)*, pages 445–456, 2008.

[196] S. Saha and S. Neogy. A case study on smart surveillance application system using WSN and IP webcam. In *Applications and Innovations in Mobile Computing (AIMoC), 2014*, 2014.

[197] Emad Felemban. Advanced border intrusion detection and surveillance using wireless sensor network technology. *International Journal of Communications, Network and System Sciences*, 6(5):251, 2013.

[198] Tian He, Sudha Krishnamurthy, John A Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *ACM International Conference on Mobile systems, applications, and services*, 2004.

[199] Ertan Onur, Cem Ersoy, Hakan Deliç, and Lale Akarun. Surveillance wireless sensor networks: deployment quality analysis. *IEEE Networking*, 21(6):48–53, 2007.

[200] Libelium. Official website. `http://www.libelium.com/`, Accessed: Apr 2016.

[201] Luis Sanchez, Luis Muoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks*, 2014.

[202] C. Pham and P. Cousin. Streaming the sound of smart cities: Experimentations on the SmartSantander test-bed. In *IEEE GreenCom and IEEE International Conference on Internet of Things and Cyber, Physical and Social Computing*, 2013.

[203] K. Daabaj, M. Dixon, and T. Koziniec. Traffic eavesdropping based scheme to deliver time-sensitive data in sensor networks. In *IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 302–308, 2010.

[204] Peter Mell, Karen Scarfone, and Sasha Romanosky. *CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0*, 2007.

[205] R. Falcon, A. Nayak, and R. Abielmona. An evolving risk management framework for wireless sensor networks. In *Conference on Computational Intelligence for Measurement Systems and Applications*, 2011.

[206] Ahmed Hasswa, Mohammad Zulkernine, and Hossam Hassanein. Routeguard: An intrusion detection and response system for mobile ad hoc networks. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications (WiMob)*, pages 336–343, 2005.

[207] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 162–175, 2004.

[208] Enrico Perla, Art Ó Catháin, Ricardo Simon Carbajo, Meriel Huggard, and Ciarán Mc Goldrick. PowerTOSSIM Z: Realistic energy modelling for wireless sensor network environments. In *Proceedings of the 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N '08, pages 35–42, New York, NY, USA, 2008. ACM.

[209] P. Levis. Collection. `http://www.tinyos.net/tinyos-2.x/doc/html/tep119.html`.

[210] Wei Dong, Yunhao Liu, Yuan He, Tong Zhu, and Chun Chen. Measurement and analysis on the packet delivery performance in a large-scale sensor network. *IEEE/ACM Transactions on Networking*, 22(6):1952–1963, December 2014.

[211] S. Sultana, G. Ghinita, E. Bertino, and M. Shehab. A lightweight secure scheme for detecting provenance forgery and packet drop attacks in wireless sensor networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2014.

[212] Ayon Chakraborty and Preethi Banala. An experimental study of jamming IEEE 802.15.4 compliant sensor networks (progress tracking). `http://www.cs.sunysb.edu/~aychakrabort/courses/cse570/`.

[213] Yongguang Zhang and Wenke Lee. Intrusion detection in wireless ad-hoc networks. In *International Conference on Mobile Computing and Networking (MobiCom)*, pages 275–283, 2000.

[214] Meng-Yen Hsieh, Yueh-Min Huang, and Han-Chieh Chao. Adaptive security design with malicious node detection in cluster-based sensor networks. *Computer Communications*, 30(11-12):2385–2400, 2007.

[215] AV. Taddeo, L. Micconi, and Alberto Ferrante. Gradual adaptation of security for sensor networks. In *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pages 1–9, 2010.

[216] Muhammad Asim, Hala Mokhtar, and Madjid Merabti. A self-managing fault management mechanism for wireless sensor networks. *CoRR*, abs/1011.5072, 2010.

[217] Sudha Krishnamurthy, Geethapriya Thamilarasu, and Christian Bauckhage. Malady: A machine learning-based autonomous decision-making system for sensor networks. In *International Conference on Computational Science and Engineering*, volume 2, pages 93–100, 2009.

[218] Mohammad Mamun, A. Kabir, Md. Hossen, and Razib Khan. Policy based intrusion detection and response system in hierarchical wsn architecture. *CoRR*, abs/1209.1678, 2012.

[219] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder policy specification language. In *International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 18–38, 2001.

[220] W3C. A p3p preference exchange language 1.0 (appel1.0). `http://www.w3.org/TR/P3P-preferences/`, 2002.

[221] OASIS. OASIS extensible access control markup language (XACML), 2005.

[222] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, C-31(1):48–59, January 1982.

[223] Sudarshan Vasudevan, Jim Kurose, and Don Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *IEEE International Conference on Network Protocols (ICNP)*, pages 350–360, 2004.

[224] Ameer Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14-15):2826–2841, 2007.

VITA

VITA

Daniele Midi

EDUCATION

- Doctor of Philosophy, September 2016
  Purdue University
  Computer Science
- Master of Science, February 2013
  University of Roma Tre (Rome, Italy)
  Computer Science Engineering, 110/110 summa cum laude
- Bachelor of Science, July 2010
  University of Roma Tre (Rome, Italy)
  Computer Science Engineering, 110/110 summa cum laude

TEACHING EXPERIENCE

- Teaching Assistant, Purdue University, CS 180 – Problem Solving and Object-Oriented Programming – Spring 2015
- Teaching Assistant, Purdue University, CS 526 – Information Security – Spring 2015